



BIOINFORMATICS

A Practical Guide to Next Generation Sequencing Data Analysis

Hamid D. Ismail



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

Bioinformatics

This book contains the latest material in the subject, covering next generation sequencing (NGS) applications and meeting the requirements of a complete semester course. This book digs deep into analysis, providing both concept and practice to satisfy the exact need of researchers seeking to understand and use NGS data reprocessing, genome assembly, variant discovery, gene profiling, epigenetics, and metagenomics. The book does not introduce the analysis pipelines in a black box, but with detailed analysis steps to provide readers with the scientific and technical backgrounds required to enable them to conduct analysis with confidence and understanding. The book is primarily designed as a companion for researchers and graduate students using sequencing data analysis but will also serve as a textbook for teachers and students in biology and bioscience.

Chapman & Hall/CRC Computational Biology Series

About the Series: This series aims to capture new developments in computational biology, as well as high-quality work summarizing or contributing to more established topics. Publishing a broad range of reference works, textbooks, and handbooks, the series is designed to appeal to students, researchers, and professionals in all areas of computational biology, including genomics, proteomics, and cancer computational biology, as well as interdisciplinary researchers involved in associated fields, such as bioinformatics and systems biology.

Metabolomics: Practical Guide to Design and Analysis

Ron Wehrens and Reza Salek

An Introduction to Systems Biology: Design Principles of Biological Circuits, Second Edition

Uri Alon

Computational Biology: A Statistical Mechanics Perspective, Second Edition

Ralf Blossey

Stochastic Modelling for Systems Biology, Third Edition

Darren J. Wilkinson

Computational Genomics with R

Altuna Akalin, Bora Uyar, Vedran Franke, and Jonathan Ronen

An Introduction to Computational Systems Biology: Systems-Level Modelling of Cellular Networks

Karthik Raman

Virus Bioinformatics

Dmitrij Frishman and Manuela Marz

Multivariate Data Integration Using R: Methods and Applications with the mixOmics Package

Kim-Anh LeCao and Zoe Marie Welham

Bioinformatics: A Practical Guide to NCBI Databases and Sequence Alignments

Hamid D. Ismail

Data Integration, Manipulation and Visualization of Phylogenetic Trees

Guangchuang Yu

Bioinformatics Methods: From Omics to Next Generation Sequencing

Shili Lin, Denise Scholtens, and Sujay Datta

For more information about this series please visit: <https://www.routledge.com/Chapman-HallCRC-Computational-Biology-Series/book-series/CRCCBS>

Bioinformatics

A Practical Guide to Next Generation Sequencing Data Analysis

Hamid D. Ismail



CRC Press

Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

First edition published 2023
by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

and by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

© 2023 Hamid D. Ismail

CRC Press is an imprint of Informa UK Limited

The right of Hamid D. Ismail to be identified as author of this work has been asserted in accordance with sections 77 and 78 of the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this book may be reprinted or reproduced or utilised in any form or by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying and recording, or in any information storage or retrieval system, without permission in writing from the publishers.

For permission to photocopy or use material electronically from this work, access www.copyright.com or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. For works that are not available on CCC please contact mpkbookspermissions@tandf.co.uk

Trademark notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Names: Ismail, Hamid D., 1967- author.

Title: Bioinformatics : a practical guide to next generation sequencing data analysis / Hamid D. Ismail.

Description: First edition. | Boca Raton : CRC Press, 2023. |

Series: Computational biology series | Includes bibliographical references and index.

Identifiers: LCCN 2022048850 (print) | LCCN 2022048851 (ebook) | ISBN 9781032409009 (hbk) |

ISBN 9781032408910 (pbk) | ISBN 9781003355205 (ebk)

Subjects: LCSH: Bioinformatics. | Genetics—Technique. | Nucleotide sequence.

Classification: LCC QH324.2.I86 2023 (print) | LCC QH324.2 (ebook) | DDC 570.285—dc23/eng/20221014

LC record available at <https://lccn.loc.gov/2022048850>

LC ebook record available at <https://lccn.loc.gov/2022048851>

ISBN: 978-1-032-40900-9 (hbk)

ISBN: 978-1-032-40891-0 (pbk)

ISBN: 978-1-003-35520-5 (ebk)

DOI: 10.1201/9781003355205

Contents

Preface, xiii

Author, xvii

CHAPTER 1 ■ Sequencing and Raw Sequence Data Quality Control	1
1.1 NUCLEIC ACIDS	1
1.2 SEQUENCING	3
1.2.1 First-Generation Sequencing	3
1.2.2 Next-Generation Sequencing	4
1.2.2.1 <i>Roche 454 Technology</i>	5
1.2.2.2 <i>Ion Torrent Technology</i>	6
1.2.2.3 <i>AB SOLiD Technology</i>	6
1.2.2.4 <i>Illumina Technology</i>	7
1.2.3 Third-Generation Sequencing	8
1.2.3.1 <i>PacBio Technology</i>	9
1.2.3.2 <i>Oxford Nanopore Technology</i>	10
1.3 SEQUENCING DEPTH AND READ QUALITY	11
1.3.1 Sequencing Depth	11
1.3.2 Base Call Quality	11
1.4 FASTQ FILES	13
1.5 FASTQ READ QUALITY ASSESSMENT	18
1.5.1 Basic Statistics	23
1.5.2 Per Base Sequence Quality	24
1.5.3 Per Tile Sequence Quality	25
1.5.4 Per Sequence Quality Scores	28
1.5.5 Per Base Sequence Content	28
1.5.6 Per Sequence GC Content	28
1.5.7 Per Base N Content	30

1.5.8	Sequence Length Distribution	30
1.5.9	Sequence Duplication Levels	31
1.5.10	Overrepresented Sequences	31
1.5.11	Adapter Content	32
1.5.12	K-mer Content	33
1.6	PREPROCESSING OF THE FASTQ READS	34
1.7	SUMMARY	45
	REFERENCES	46
<hr/> CHAPTER 2 ■ Mapping of Sequence Reads to the Reference Genomes		49
2.1	INTRODUCTION TO SEQUENCE MAPPING	49
2.2	READ MAPPING	55
2.2.1	Trie	56
2.2.2	Suffix Tree	56
2.2.3	Suffix Arrays	57
2.2.4	Burrows–Wheeler Transform	58
2.2.5	FM-Index	62
2.3	READ SEQUENCE ALIGNMENT AND ALIGNERS	63
2.3.1	SAM and BAM File Formats	65
2.3.2	Read Aligners	70
2.3.2.1	<i>Burrows–Wheeler Aligner</i>	71
2.3.2.2	<i>Bowtie2</i>	75
2.3.2.3	<i>STAR</i>	76
2.4	MANIPULATING ALIGNMENTS IN SAM/BAM FILES	79
2.4.1	Samtools	79
2.4.1.1	<i>SAM/BAM Format Conversion</i>	79
2.4.1.2	<i>Sorting Alignment</i>	80
2.4.1.3	<i>Indexing BAM File</i>	80
2.4.1.4	<i>Extracting Alignments of a Chromosome</i>	81
2.4.1.5	<i>Filtering and Counting Alignment in SAM/BAM Files</i>	81
2.4.1.6	<i>Removing Duplicate Reads</i>	82
2.4.1.7	<i>Descriptive Statistics</i>	83
2.5	REFERENCE-GUIDED GENOME ASSEMBLY	83
2.6	SUMMARY	85
	REFERENCES	86

CHAPTER 3 ■ De Novo Genome Assembly	89
3.1 INTRODUCTION TO DE NOVO GENOME ASSEMBLY	89
3.1.1 Greedy Algorithm	90
3.1.2 Overlap-Consensus Graphs	90
3.1.3 De Bruijn Graphs	91
3.2 EXAMPLES OF DE NOVO ASSEMBLERS	93
3.2.1 ABySS	93
3.2.2 SPAdes	97
3.3 GENOME ASSEMBLY QUALITY ASSESSMENT	99
3.3.1 Statistical Assessment for Genome Assembly	100
3.3.2 Evolutionary Assessment for De Novo Genome Assembly	103
3.4 SUMMARY	106
REFERENCES	107
CHAPTER 4 ■ Variant Discovery	109
4.1 INTRODUCTION TO GENETIC VARIATIONS	109
4.1.1 VCF File Format	110
4.1.2 Variant Calling and Analysis	113
4.2 VARIANT CALLING PROGRAMS	114
4.2.1 Consensus-Based Variant Callers	114
4.2.1.1 <i>BCF Tools Variant Calling Pipeline</i>	115
4.2.2 Haplotype-Based Variant Callers	125
4.2.2.1 <i>FreeBayes Variant Calling Pipeline</i>	127
4.2.2.2 <i>GATK Variant Calling Pipeline</i>	129
4.3 VISUALIZING VARIANTS	143
4.4 VARIANT ANNOTATION AND PRIORITIZATION	143
4.4.1 SIFT	145
4.4.2 SnpEff	148
4.3.3 ANNOVAR	151
4.3.3.1 <i>Annotation Databases</i>	153
4.3.3.2 <i>ANNOVAR Input Files</i>	156
4.5 SUMMARY	160
REFERENCES	161

CHAPTER 5 ■ RNA-Seq Data Analysis	163
5.1 INTRODUCTION TO RNA-SEQ	163
5.2 RNA-SEQ APPLICATIONS	165
5.3 RNA-SEQ DATA ANALYSIS WORKFLOW	166
5.3.1 Acquiring RNA-Seq Data	166
5.3.2 Read Mapping	167
5.3.3 Alignment Quality Assessment	171
5.3.4 Quantification	172
5.3.5 Normalization	174
5.3.5.1 <i>RPKM and FPKM</i>	174
5.3.5.2 <i>Transcripts per Million</i>	175
5.3.5.3 <i>Counts per Million Mapped Reads</i>	175
5.3.5.4 <i>Trimmed Mean of M-values</i>	175
5.3.5.5 <i>Relative Expression</i>	176
5.3.5.6 <i>Upper Quartile</i>	176
5.3.6 Differential Expression Analysis	176
5.3.7 Using EdgeR for Differential Analysis	180
5.3.7.1 <i>Data Preparation</i>	181
5.3.7.2 <i>Annotation</i>	183
5.3.7.3 <i>Design Matrix</i>	184
5.3.7.4 <i>Filtering Low-Expressed Genes</i>	185
5.3.7.5 <i>Normalization</i>	186
5.3.7.6 <i>Estimating Dispersions</i>	186
5.3.7.7 <i>Exploring the Data</i>	189
5.3.7.8 <i>Model Fitting</i>	194
5.3.7.9 <i>Ontology and Pathways</i>	202
5.3.8 Visualizing RNA-Seq Data	204
5.3.8.1 <i>Visualizing Distribution with Boxplots</i>	206
5.3.8.2 <i>Scatter Plot</i>	207
5.3.8.3 <i>Mean-Average Plot (MA Plot)</i>	208
5.3.8.4 <i>Volcano Plots</i>	209
5.4 SUMMARY	209
REFERENCES	211

CHAPTER 6 ■ Chromatin Immunoprecipitation Sequencing	213
6.1 INTRODUCTION TO CHROMATIN IMMUNOPRECIPITATION	213
6.2 CHIP SEQUENCING	214
6.3 CHIP-SEQ ANALYSIS WORKFLOW	215
6.3.1 Downloading the Raw Data	217
6.3.2 Quality Control	218
6.3.3 ChIP-Seq and Input Read Mapping	219
6.3.4 ChIP-Seq Peak Calling with MACS3	223
6.3.5 Visualizing ChIP-Seq Enrichment in Genome Browser	226
6.3.6 Visualizing Peaks Distribution	229
6.3.6.1 <i>ChIP-Seq Peaks' Coverage Plot</i>	230
6.3.6.2 <i>Distribution of Peaks in Transcription Start Site (TSS) Regions</i>	233
6.3.6.3 <i>Profile of Peaks along Gene Regions</i>	234
6.3.7 Peak Annotation	235
6.3.7.1 <i>Writing Annotations to Files</i>	237
6.3.8 ChIP-Seq Functional Analysis	239
6.3.9 Motif Discovery	243
6.4 SUMMARY	250
REFERENCES	251
CHAPTER 7 ■ Targeted Gene Metagenomic Data Analysis	253
7.1. INTRODUCTION TO METAGENOMICS	253
7.2 ANALYSIS WORKFLOW	254
7.2.1 Raw Data Preprocessing	254
7.2.2 Metagenomic Features	255
7.2.2.1 <i>Clustering</i>	255
7.2.2.2 <i>Denoising</i>	256
7.2.3 Taxonomy Assignment	258
7.2.3.1 <i>Basic Local Alignment Search Tool</i>	258
7.2.3.2 <i>VSEARCH</i>	259
7.2.3.3 <i>Ribosomal Database Project</i>	259
7.2.4 Construction of Phylogenetic Trees	260
7.2.5 Microbial Diversity Analysis	261
7.2.5.1 <i>Alpha Diversity Indices</i>	262
7.2.5.2 <i>Beta Diversity</i>	262

7.3	DATA ANALYSIS WITH QIIME2	263
7.3.1	QIIME2 Input Files	265
7.3.1.1	<i>Importing Sequence Data</i>	265
7.3.1.2	<i>Metadata</i>	269
7.3.2	Demultiplexing	269
7.3.3	Downloading and Preparing the Example Data	271
7.3.3.1	<i>Downloading the Raw Data</i>	271
7.3.3.2	<i>Creating the Sample Metadata File</i>	272
7.3.3.3	<i>Importing Microbiome Yoga Data</i>	274
7.3.4	Raw Data Preprocessing	275
7.3.4.1	<i>Quality Assessment and Quality Control</i>	275
7.3.4.2	<i>Clustering and Denoising</i>	278
7.3.5	Taxonomic Assignment with QIIME2	289
7.3.5.1	<i>Using Alignment-Based Classifiers</i>	289
7.3.5.2	<i>Using Machine Learning Classifiers</i>	291
7.3.6	Construction of Phylogenetic Tree	297
7.3.6.1	<i>De Novo Phylogenetic Tree</i>	297
7.3.6.2	<i>Fragment-Insertion Phylogenetic Tree</i>	298
7.3.7	Alpha and Beta Diversity Analysis	298
7.4	SUMMARY	300
REFERENCES		301
<hr/> CHAPTER 8 ■ Shotgun Metagenomic Data Analysis		303
8.1	INTRODUCTION	303
8.2	SHOTGUN METAGENOMIC ANALYSIS WORKFLOW	305
8.2.1	Data Acquisition	305
8.2.2	Quality Assessment and Processing	305
8.2.3	Removing Host DNA Reads	306
8.2.3.1	<i>Download Human Reference Genome</i>	306
8.2.3.2	<i>Mapping Reads to the Reference Genome</i>	307
8.2.3.3	<i>Converting SAM to BAM Format</i>	307
8.2.3.4	<i>Separating Metagenomic Reads in BAM Files</i>	307
8.2.3.5	<i>Creating Paired-End FASTQ Files from BAM Files</i>	308
8.2.4	Assembly-Free Taxonomic Profiling	310
8.2.4	Assembly of Metagenomes	315

8.2.5	Assembly Evaluation	317
8.2.6	Mapping Reads to the Assemblies	318
8.2.7	Binning	321
8.2.8	Bin Evaluation	323
8.2.9	Prediction of Protein-Coding Region	324
8.3	SUMMARY	325
	REFERENCES	326
	INDEX, 327	



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Preface

THE USE OF NEXT-GENERATION sequencing data analysis is the only analysis that can make sense of the massive genomic data produced by the high-throughput sequencing technologies and accumulated in gigabytes and terabytes in our hard drives and cloud databases. With the presence of computational resources and elegant algorithms for NGS data analysis, scientists need to know how to master the tools of these analyses to achieve the goals of their research. Learning NGS data analysis techniques has already become one of the most important assets that bioinformaticians and biologists must acquire to keep abreast of the progress in the modern biology and to avail of the genomic technologies and resources that have become the *de facto* in bioscience research and applications including diagnosis, drug and vaccine discovery, medical studies, and the investigations of pathways that give clues to many biological activities and pathogenicity of diseases.

In the last two decades, the progress of next-generation sequencing has made a strong positive impact on human life and a forward stride in human civilization. Introduction of new sequencing technologies revolutionizes the bioscience. As a result, a new field of biology called genomics has emerged. Genomics focuses on the composition, structure, functional units, evolution, and manipulation of genomes, and it generates massive amount of data that need to be ingested and analyzed. As a consequence, bioinformatics has also emerged as an interdisciplinary field of science to address the specific needs in data acquisition, storage, processing, analysis, and integration of that data into a broad pool to enrich the genomic research.

This book is designed primarily to be a companion for the researchers and graduate students who use sequencing data analysis in their research, and it also serves as a textbook for teachers and students in biology and bioscience. It contains an updated material in the subject covering most NGS applications and meeting the requirements of a complete semester course. The reader will find that this book is digging deep in the analysis, providing both concept and practice to satisfy the exact need of the researchers who seek to understand and use NGS data reprocessing, genome assembly, variant discovery, gene profiling, epigenetics, and metagenomics. The book does not introduce the analysis pipelines in a black box as the existing books do, but with the analysis steps, it pervades each topic in detail to provide the readers with the scientific and technical background that enable them to conduct the analysis with confidence and understanding.

The book consists of eight chapters. All chapters include real-world worked examples that demonstrate the steps of the analysis workflow with real data downloadable from the

databases. Most programs used in this book are open-source Unix/Linux-based programs. Others can be used in Anaconda environments.

Chapter 1 discusses sequencing data acquisition from NGS technologies and databases, FASTQ file format, and Phred base call quality. The chapter covers the quality assessment of the FASTQ and read quality metrics in some detail so that the readers can diagnose potential problems in raw data and learn how to fix any possible quality problem before analysis.

Chapter 2 discusses read alignment/mapping to reference genomes. The strategies of both reference genome indexing algorithms and read mapping algorithms are discussed in detail with illustrations so that the readers can understand how mapping process works, the different indexing and alignment algorithms currently used, and which aligners are good for RNA sequencing applications. The chapter discusses indexing and searching algorithms like suffix tree, suffix arrays, Burrow-Wheeler Transform (BWT), FM-index, and hashing, which are the algorithms used by aligners. The chapter then discusses the mapping process and aligners like BWA, Bowtie, STAR, etc. The SAM/BAM file format is discussed in detail so that the reader can understand how alignment information are stored in fields in the SAM/BAM file. Finally, the chapter discusses the manipulation of alignments in SAM/BAM files using Samtools programs for different purposes, including SAM to BAM conversion, alignment sorting, indexing BAM files, extracting alignments of a chromosome or a specific region, filtering and counting alignment, removing duplicate reads, and generating descriptive statistics.

Chapter 3 discusses de novo genome assembly and de novo assembly algorithms including greedy algorithm, overlap-consensus graphs, and de Bruijn graphs. The quality assessment of the assembled genome is discussed through two approaches: statistical approach and evolutionary approach.

Chapter 4 covers variant calling (SNPs and InDels) in detail. The introduction of this chapter discusses variants, variant file format (VCF), and the general workflow of the variant calling. The chapter then discusses both consensus-based variant calling and haplotype-based variant calling and example callers from each group including BCFTools, FreeBayes, and GATK best practice variant calling pipelines. Finally, the chapter discusses variant annotation and prioritization and annotation programs including SIFT, SnpEff, and ANNOVAR.

Chapter 5 discusses RNA-Seq data analysis. The introduction includes RNA-Seq basics and applications. The chapter then discusses the steps of RNA-Seq analysis workflow, including data acquisition, read alignment, alignment quality control, quantification, RNA-Seq data normalization, statistical modeling and differential expression analysis, using R packages for differential analysis, and visualization of RNA-Seq data.

Chapter 6 covers ChIP-Seq data analysis. It discusses in detail the workflow of ChIP-Seq data analysis including data acquisition, quality control, read mapping, peak calling, visualizing peak enrichment and peak distribution, peak annotation, peak functional analysis, and motif discovery.

Chapter 7 discusses targeted gene metagenomic data analysis (amplicon-based microbial analysis) for environmental and clinical samples. The chapter covers raw data preprocessing

(demultiplexing and quality control), metagenomic feature processing (OTU clustering, DADA2 denoising, Deblur denoising, and UNOISE2 denoising), taxonomy assignment and construction of phylogenetic trees, microbial diversity analysis (alpha diversity and beta diversity), and data analysis with QIIME2.

Chapter 8 discusses the shotgun metagenomic data analysis. The discussion includes data acquisition, quality control, removal of host DNA, assembly-free taxonomic profiling, assembly evaluation, *de novo* assembly of metagenomes, binning, bin evaluation, and prediction of gene transcript and gene annotation.

The computer codes used in the book chapters and their updates will be available at “<https://github.com/hamiddi/ngs>”.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Author

Hamid D. Ismail, PhD, earned an MSc and a PhD in computational science at North Carolina Agricultural and Technical State University (NC A&T), USA, and DVM and BSc at the University of Khartoum, Sudan. He also earned several professional certifications, including SAS Advanced Programmer and SQL Expert Programmer. Currently, he is a postdoc scholar at Michigan Technological University and an adjunct professor of data science and bioinformatics at NC A&T State University. Dr. Ismail is a bioinformatician, biologist, data scientist, statistician, and machine learning specialist. He has contributed widely to the field of bioinformatics by developing bioinformatics tools and methods for applications of machine learning on genomic data.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Sequencing and Raw Sequence Data Quality Control

1.1 NUCLEIC ACIDS

Nucleic acids are the chemical molecules that every living organism must have. They carry information that directs biological activities in cells and determines the inherited characteristics of the living organism. The two main kinds of nucleic acids are deoxyribonucleic acid (DNA) and ribonucleic acid (RNA). DNA is the master blueprint for life or the book of life, and it constitutes the genetic material in prokaryotic and eukaryotic cells and virions. The RNA is the main genetic material of the RNA viruses, but it is found in other organisms as molecules transcribed by DNA to play important biological roles such as protein synthesis and gene regulation. The set of the DNA particles in both prokaryotic and eukaryotic cells is called the genome. RNA is the genome of only some viruses (RNA viruses). A nucleic acid (DNA/RNA) is a polymer made up of four building blocks called nucleotides. A molecule of the nucleotide consists of (i) a sugar molecule (either deoxyribose in DNA or ribose in RNA) attached to a phosphate group and (ii) a nitrogen-containing base called nucleobase. In general, the nucleic acid sequence is made up of four nucleotides distinguished from one another only by the nitrogen-containing bases (Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) in the DNA molecule and Adenine (A), Cytosine (C), Guanine (G), and Uracil (U) in the RNA molecule). Those four nucleobases are divided into pyrimidine and purine bases. Pyrimidine bases include cytosine, thymine, and uracil; they are aromatic heterocyclic organic compound with a single ring. Purine bases include adenine and guanine which have two heterocyclic ring structures. A DNA molecule exists in the form of two complementary strands (forward and reverse) that wind around each other forming a double-helix structure. The two strands are held together by hydrogen bonds formed between the bases (adenine is a base pair of thymine (A/T), and cytosine is a

base pair of guanine (C/G)) as shown in Figure 1.1. Adenine and thymine form two hydrogen bonds (weak bond), while cytosine and guanine form three hydrogen bonds (strong bond). Those base pairings are specific so that a sequence of a strand is predicted from the other one. The length of a DNA sequence is given in base pair (bp), kilobase pair (kbp), or megabase pair (Mbp). The RNA exists in a single strand; however, it sometimes forms double-stranded secondary structure with itself to perform specific function.

The genome of an organism is the book of life for that organism. It determines the living aspects and biological activities of cells. A genome contains coding regions known as genes that carry information for protein synthesis. Genes are transcribed into messenger RNA (mRNA), which is translated into proteins and the proteins control most of the biological processes in the living organisms.

A gene consists of coding regions, non-coding regions, and a regulatory region. The coding regions in the eukaryotic genes are not continuous, but non-coding sequences (called introns) are found between the coding sequences (called exons). These introns are removed from the transcribed transcripts before protein translation, leaving only the exons which form the coding region called the open reading frame (**ORF**). Each eukaryotic gene has its own regulatory region that controls its expression. In prokaryotic cells, a group of genes, called an operon, are regulated by a single regulatory region. The viruses, which fall in the margin between living organisms and chemical particles, function and replicate only inside host cells by using the host cells machineries such as ribosomes to create structural and non-structural proteins of viruses and to replicate to create new virions.

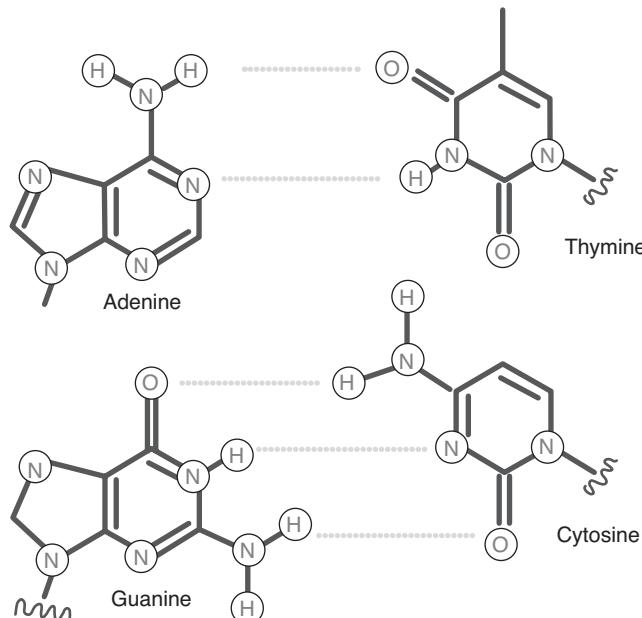


FIGURE 1.1 Base pairing and hydrogen bonds between pairs of the DNA nucleotides.

1.2 SEQUENCING

DNA/RNA sequencing is the determination of the order of the four nucleotides in a nucleic acid molecule. The recovered order of the nucleotides in a genome of an organism is called a sequence. Sequencing of the DNA helps scientists to investigate the functions of genes, roles of mutations in traits and diseases, species, evolutionary relationships between species, diagnosis of diseases caused by genetic factors, development of gene therapy, criminal investigations and legal problems, and more. Since the nucleotides are distinguished by the bases, the DNA and RNA sequences are represented in bioinformatics by the sequences of the four-nucleobase single-character symbols (A, C, G, and T for DNA and A, C, G, and U for RNA).

The attempts to sequence nucleic acid began immediately after the landmark discovery in 1953 of the double-helix structure of the DNA by James Watson and Francis Crick. The alanine tRNA was the first nucleic acid sequenced in 1965 by the Nobel prize winner Robert Holley. Holley used two ribonuclease enzymes to split the tRNA at specific nucleotide positions, and the order of the nucleotides was determined manually [1]. The first DNA molecule was sequenced in 1972 by Walter Fiers. That DNA molecule was the gene that codes the coat protein of the bacteriophage MS2, and the sequencing was made by using enzymes to break the bacteriophage RNA into pieces and separating the fragments with electrophoresis and chromatography [2]. The sequencing of the alanine tRNA by Robert Holley and the sequencing of the gene of the bacteriophage MSE coat protein are ones of the major milestones in the history of genomics and DNA sequencing. They paved the way for the first-generation sequencing.

1.2.1 First-Generation Sequencing

The early 1970s witnessed the emergence of the first-generation sequencing when the American biologists Allan M. Maxam and Walter Gilbert developed a chemical method for sequencing, followed by the English biochemist Frederick Sanger who developed the chain-terminator method. The Sanger method became the more commonly used first-generation sequencing method to this date. Both methods were used in the shotgun sequencing, which involves breaking genome into DNA fragments and sequencing of the fragments individually. The genome sequence is then assembled based on the overlaps after aligning the fragment sequences.

The Maxam–Gilbert sequencing method is based on the chemical modification of DNA molecules and subsequent cleavage at specific bases. In the Maxam–Gilbert sequencing method, first, the DNA is denatured (separation of the DNA strands) by heating or helicase enzyme into single-stranded DNA (ssDNA) molecules. The ssDNA is run in the gel electrophoresis to separate the two DNA strands into two bands. Any one of the bands (strand) can be cut from the gel and sequenced. In the sequencing step, the solution with ssDNA is then divided into four reaction tubes labeled A+G, G, C+T, and C. The ssDNA in each tube is labeled chemically with an isotope and treated with a specific chemical that breaks the DNA strand at a specific nucleotide according to the tube labels. After the reaction, polyacrylamide gel is then used for running the four reactions in four separate lanes (A+G, G,

C+T, and C). The order of the nucleotides (A, C, G, and T) in the DNA sequence can then be solved from the bands on the gel.

On the other hand, the steps of the Sanger sequencing method are similar to that of the polymerase chain reaction (PCR) including denaturing, primer annealing, and complementary strand synthesis by polymerase. However, in the Sanger sequencing, the sample DNA is divided into four reaction tubes labeled ddATP, ddGTP, ddCTP, and ddTTP. In the four reaction tubes, the four types of deoxynucleotides triphosphates (dATP, dGTP, dCTP, and dTTP) are added as in the PCR but one of the four radio-labeled dideoxynucleotide triphosphates (ddATP, ddGTP, ddCTP, or ddTTP) is also added to the reactions, as labeled, to terminate the DNA synthesis at certain positions of known nucleotides. The synthesis termination results in DNA fragments of varying lengths ending with the labeled ddNTPs. Those fragments are then separated by size using gel electrophoresis on a denaturing polyacrylamide-urea gel with each of the four reactions running in a separate lane labeled A, T, G, and C. The DNA fragments will be separated by lengths; the smaller fragments will move faster in the gel. The DNA bands are then graphed by autoradiography, and the order of the nucleotide bases on the DNA sequence can be directly read from the X-ray film or the gel image.

1.2.2 Next-Generation Sequencing

The next-generation sequencing (NGS) was invented a few decades after the invention of the first-generation sequencing. Unlike the first-generation sequencing, NGS produces massive number of sequences from a single sample in a short period of time, with lower costs, and it can process multiple samples simultaneously. Millions to billions of DNA nucleotides are sequenced in parallel, yielding substantially massive sequences. With the NGS, millions of prokaryotic, eukaryotic, and viral genomes were sequenced. Rather than chain termination, the NGS uses library or fragmented DNA to solve the order of the nucleotide in a targeted sequence. The NGS is used in many applications including the sequencing of the whole genome, whole transcriptome, targeted genes or transcripts, sequencing of the genomic regions where the epigenetic modifications or protein interactions take place. Hence, the NGS can be used for genome assembly, mutation or variant discovery, gene expression studies, epigenetics, and metagenomics. Those applications are discussed in detail in the next chapters.

After DNA or RNA sample collection, the step of the NGS process is the library preparation in which the sequencing libraries are constructed for the DNA or RNA sample of interest. The RNA is converted into complementary DNA (cDNA) before library preparation. The library preparation involves breaking the DNA into small fragments (fragmentation step) using sonication or enzymes. The size of the fragment can be adjusted to a specific length or range. The fragmentation is followed by repairing or blunting the ends of the fragments which have unpaired or overhanging nucleotides (end-repair step). The next step is the ligation of the adaptors to the ends of the DNA fragments. The adaptors are artificially synthesized sequences that include certain parts to serve specific purposes. The free end of a ligated adaptor is made of an anchoring sequence that can attach to the surface of the flow cell slide where sequencing takes place. The adaptor also includes universal primers

for PCR DNA strand synthesis and barcode sequence for indexing the sample DNA. This allows multiple samples (multiplexing) to be sequenced in a single run; the DNA fragments of each sample will have a unique barcode. Later, after sequencing, the sample sequences can be separated in the analysis by demultiplexing. In the sequencing of some application like gene expression (RNA-Seq) and epigenetics, an enrichment step is usually included to amplify or to separate only the targeted sequences. In RNA application, enrichment is performed to separate mRNA from the other types of RNA. In epigenetics, the genomic regions where protein interaction is taking place can also be enriched. The enrichment is usually performed with PCR, but there are other means as well. The library preparation of the DNA/RNA is similar for all NGS technologies but the sequencing process is different from one to another. The sequences produced by the NGS technologies range between 75 and 400 base pairs (bp) in length. These sequences are called short reads. In general, short-read sequencing (SRS) can either be a single-end sequencing, which sequences the forward strand only, or paired-end sequencing, which sequences both forward and reverse strands. The latter reduces the chance of making basal error in the resulted sequence. The DNA or RNA reads consist of the four nucleobase characters A, C, G, and T. However, the sequence may also include N for an unresolved base.

1.2.2.1 Roche 454 Technology

Roche 454 pioneered the NGS, when 454 was used to sequence the whole genome of *Mycoplasma genitalium* in 2006 [3]. The 454 technology uses pyrosequencing, which depends on the sequential addition and incorporation of nucleotides in the DNA template. The signal of the added nucleotide is quantitated by conversion of released pyrophosphate into a light signal in the real time. The pyrosequencing is based on a series of enzymatic reactions that lead to the DNA synthesis and release of the inorganic pyrophosphate every time a nucleotide is incorporated by polymerase in the DNA chain. The density of the light generated by the reactions can be detected by a charge-coupled device camera. The order of the nucleotides in the DNA template is determined by quantitating the light density. In the pyrosequencing, the DNA is fragmented and denatured into ssDNA. Two adaptors (A and B) are ligated to both ends of the fragments. Beads of soluble particles with single-stranded primers complementing adaptor A are added to the reactions. The adaptor A attached to ssDNA template complements the bead primers, which initiate the synthesis of the complementary strand. This step can be repeated several times for enrichment (PCR). Then, the beads with the ssDNA templates are placed into wells where sequencing takes place. A primer is added to complement the adaptor B and to initiate the addition of new nucleotides to the complementary strand. However, this time, known nucleotides are added. Every time a nucleotide is incorporated into the complementary strand, a hydroxyl group of the last nucleotide reacts with the alpha phosphate of the incorporated nucleotide releasing a two-phosphate compound called the inorganic pyrophosphate (PPI). The PPI contains a high amount of energy that converts the adenosine monophosphate (AMP) into adenosine triphosphate (ATP) with the help of ammonium persulfate (APS) and sulfurylase which are added to the reaction. Finally, luciferin and luciferase are added to the ATP so the luciferin forms light. Every time a nucleotide is added, a light is emitted

and captured by a sensor or a camera. Figure 1.2 shows the steps of pyrosequencing after adding the DNA fragments to the wells, where sequencing takes place by incorporating a known nucleotide to the growing complementary strand each time and releasing of PPI that is translated into a light signal, which is detected by a camera.

1.2.2.2 Ion Torrent Technology

The Ion Torrent sequencers are manufactured by Thermo Fisher Scientific. The Ion Torrent sequencing method is similar to the Roche 454 sequencing in the first steps: fragmentation, DNA denaturing, adaptor ligation, and the use of beads in microwells with primers. However, instead of solving the nucleotide order of the DNA template by capturing the light released by inorganic pyrophosphate as in Roche 454 sequencing, the Ion Torrent method depends on the pH signal of the hydrogen ions released every time a nucleotide is incorporated in the DNA template. The microwells are in a microchip provided with the so-called ion sensitive layer (pH sensor). This layer is able to detect the hydrogen ions (protons). There are four solutions for the four kinds of nucleotides (dATP, dCTP, dGTP, and dTTP). During the Ion Torrent sequencing, the four nucleotide solutions are added sequentially. If the nucleotide is incorporated, a hydrogen ion (H^+) will be released changing the pH of the solution and a signal will be generated; otherwise, no change in the pH or a signal (Figure 1.3). The base call is based on this signal. Each time, the microwells are washed to remove the free nucleotides before adding a new nucleotide solution.

1.2.2.3 AB SOLiD Technology

The Applied Biosystem SOLiD is based on the sequential ligation with dye-labeled oligos. SOLiD stands for Sequencing by Oligonucleotide Ligation and Detection. This technology was developed by Fisher Scientific. It depends on the DNA ligase that has the ability to incorporate nucleotides in the DNA template. The DNA library preparation is similar to

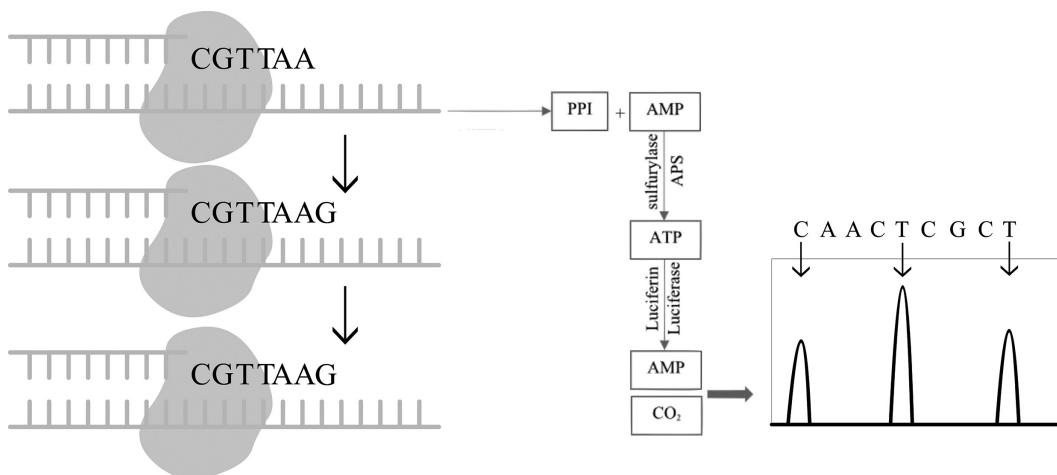


FIGURE 1.2 Pyrosequencing and base calling.

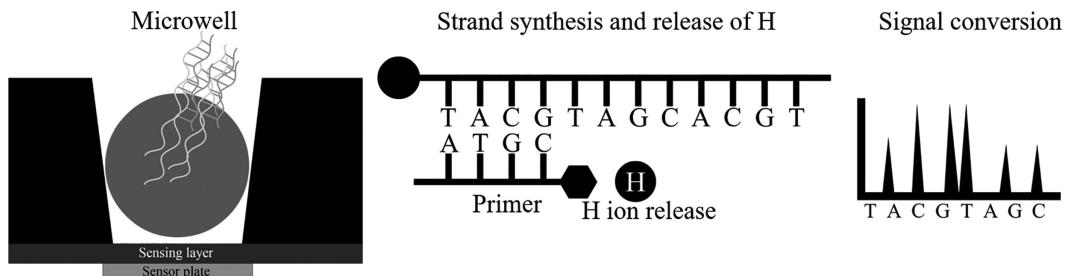


FIGURE 1.3 Ion Torrent sequencing.

what has been discussed above. In the library preparation step, the DNA is broken down into fragments. The ends of these fragments are repaired and then adaptor sequences are ligated to the ends. As Roche 454 and Ion Torrent sequencing, beads are also used and the fragments attached to the beads are amplified using PCR. After the amplification, specific primers are used to synthesize strands complementing the ssDNA templates and providing a free 5' phosphate group (instead of 3' hydroxyl group) that can ligate to one of a set of fluorescently labeled probes. A probe is an 8-mer oligonucleotide with a fluorescent dye at the 5'-end and a hydroxyl group at the 3'-end. The first two nucleotides of the probe specifically complement the nucleotides on the sequenced fragments. The next three nucleotides are universal that can bind to any one of the four nucleotides. The remaining three nucleotides of the probe (at the 5'-end) are also universal but are fluorescently labeled and they are able to be cleaved during the sequencing leaving only the other five nucleotides binding to the template strand. The set of probes consists of a combination of 16 possible 2-nucleotides that can ligate to the primer sequence on the fragments. Every time two specific nucleotides ligate, the last three nucleotides of the probe are cleaved and the fluorescence specific to the ligated probe is emitted, captured, and translated into the corresponding two bases. This step is followed by removing the fluorescently labeled nucleotide and regenerating the 5' phosphate group. The process (ligation, detection, and cleavage) is repeated multiple times for extending the complementary strand. The strand extension products are then removed, and a new primer is used for the second round of ligation, detection, and cleavage. The cycles are then repeated and every time a new primer is used for the remaining of the DNA fragments.

1.2.2.4 Illumina Technology

Illumina uses sequencing by synthesis (SBS) approach (Figure 1.4), in which it uses four fluorescently labeled nucleotides to sequence the DNA fragments on the flow cell surface in multiple cycles. In each cycle, a single fluorescently labeled nucleotide (dATP, dCTP, dGTP, or dTTP) is added to the DNA growing chain. These labeled nucleotides serve as chain terminators that stop chain extension and also the incorporation of the labeled nucleotide in the chain causes color emission or a signal of a specific nucleotide. The signal of the incorporated nucleotide is then captured by an imaging camera to identify that nucleotide (or base). The terminator nucleotide is then cleaved to allow the incorporation of the next

nucleotide. The base call is based on the intensity of the signals during chain synthesis and labeled nucleotide incorporation.

The DNA library preparation, as described above, includes the DNA fragmentation, DNA end repair, and adaptor ligation. Sequencing is carried out on the solid surface of a flow cell divided into lanes. On the surface, there are two types of oligonucleotides that complement the anchor sequence in the adaptors attached to the DNA template. When the DNA fragments are added to the cell flow, the anchor sequences in the fragments anneal to (complement) the surface oligonucleotides forming bridges. The oligos also act as primers that initiate the synthesis of complementary strands generating clusters of the DNA fragments. The sequencing step begins by denaturing the DNA strands and adding fluorescently labeled nucleotides. Only one nucleotide is incorporated in the DNA template at a time. After the addition of each nucleotide, the clusters are excited by a light source and a signal is emitted. The signal intensity determines the base call.

1.2.3 Third-Generation Sequencing

The third-generation sequencing (TGS) is relatively new sequencing technology developed as a result of the need for long reads to improve the accuracy and resolution of sequencing

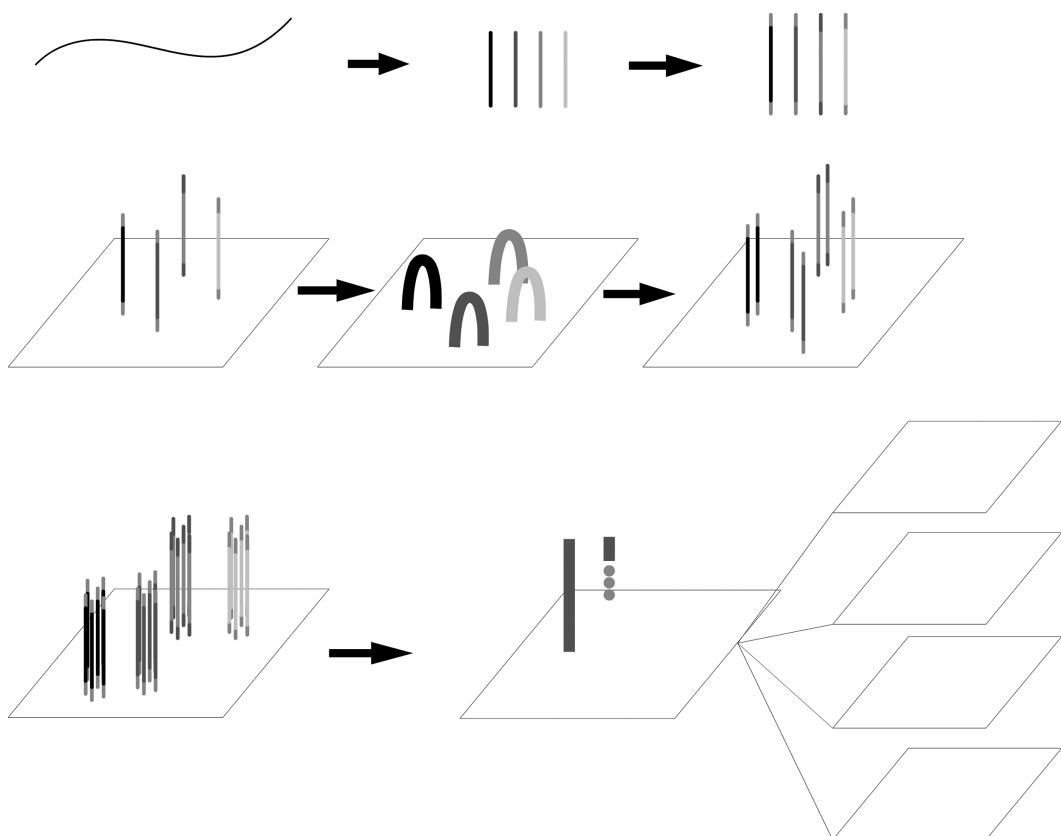


FIGURE 1.4 Illumina sequencing.

applications like the de novo genome assembly, variant discovery, and epigenetics. Usually, genomes have long repeated sequences ranging from hundreds to thousands of bases that are hard to cover with short reads produced by the NGS. The foundation for TGS was emerged in 2003, when the DNA polymerase was used to obtain a sequence of 5 bp from a single DNA molecule by using fluorescent microscopy [4]. The single-molecule sequencing (SMS) then evolved to include (i) direct imaging of individual DNA molecules using advanced microscopy techniques and (ii) nanopore sequencing technologies in which a single molecule of DNA is threaded through a nanopore and molecule bases are detected as they pass through the nanopore. Although TGS provides long reads (from a few hundred to thousands of base pairs), that may come at the expense of the accuracy. However, lately, the accuracy of the TGS has been greatly improved. The TGS provides long reads that can enhance de novo assembly and enable direct detection of haplotypes and higher consensus accuracy for better variant discovery. In general, there are two TGS technologies that are currently available: (i) Pacific Bioscience (PacBio) single-molecule real-time (SMRT) sequencing and (ii) Oxford Nanopore Technologies (ONTs).

1.2.3.1 *PacBio Technology*

The Pacific Biosciences (PacBio) sequencing can provide long reads that range between 500 and 50,000 bp. The PacBio sequencing has been improved since it made debut in 2011. The underlying technology of the PacBio is based on the SMRT sequencing, in which a single DNA molecule is sequenced and the base calling is given in the real time, while the sequencing is in progress [5]. The sequencing steps include fragmentation and ligation of adaptors to the DNA template for library generation. Special loop adaptors are ligated to the ssDNA produced from the double-stranded DNA (dsDNA). The loop adaptors link both strands forming structures called linear DNA SMRTbells. The sequencing takes place on nano wells on a flow cell. The nano wells are made of silicon dioxide chips called zero-mode waveguides (ZMWs) [6]. A cell contains thousands of ZMWs. A ZMW is around 70 nm in diameter and 100 nm in depth, and it allows laser light to come through the bottom to excite the fluorescent dye. A DNA polymerase is attached to the bottom of the nano well. When a DNA single fragment is added to the well, the DNA polymerase is attached to it. The polymerase has the strand displacement capability that converts the DNA SMRTbells into circular structure called circular DNA SMRTbell (Figure 1.5). Then, the DNA polymerase continues adding nucleotides to form a complementary strand for both forward and reverse strands. PacBio uses SBS approach. Four fluorescently labeled nucleotides (dNTPs) are added to the reactions. The dNTPs are fluorescently labeled by attaching the fluorescent dyes to phosphate chain of the nucleotides. Each time a fluorescently labeled nucleotide is incorporated, a fluorescent dye is cleaved from the growing nucleic acid chain before the next nucleotide is added. The fluorescence is then excited by the light coming through the bottom of the well and detected in the real time. The real-time identification of the incorporated labeled nucleotides allows the base call. This process of adding nucleotides and fluorescence detection continues until the entire fragment is sequenced. This pass can be repeated different times to generate more accurate reads by the circular consensus sequences (CCS). Ten passes produce reads with 99.9% accuracy. These reads are

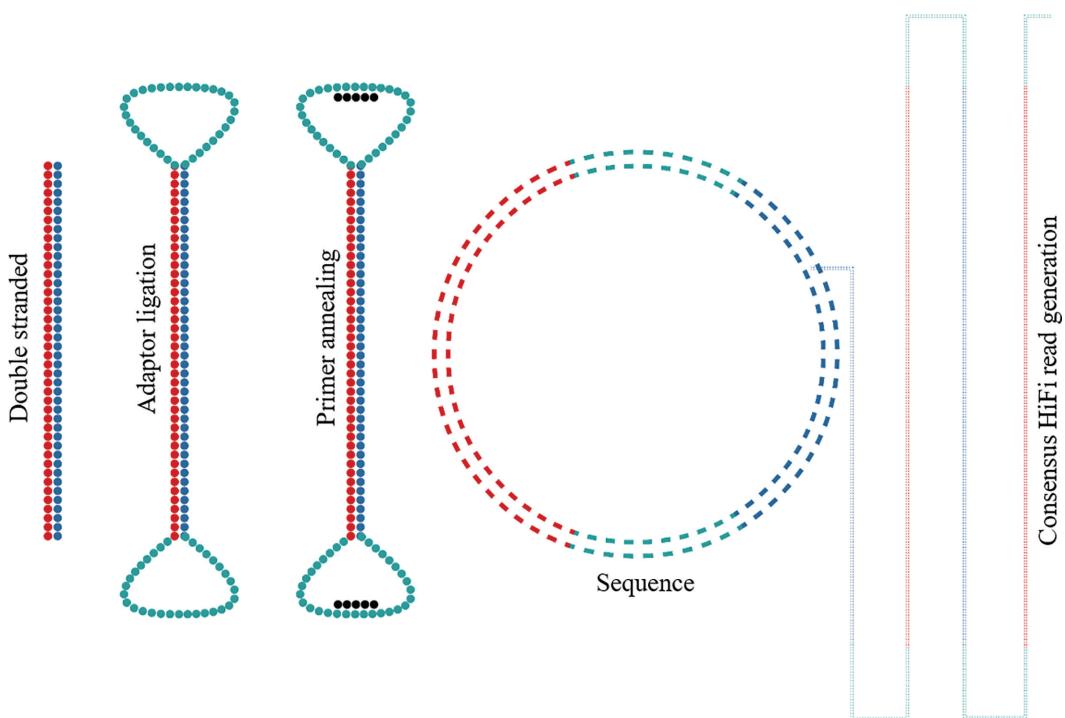


FIGURE 1.5 Pacific Bioscience (PacBio) sequencing.

called highly accurate long sequencing reads (HiFi reads). The sequencing is carried out in parallel on thousands of nanophotonic wells, generating long reads. PacBio can produce long reads with 99% accuracy (>Q20) and uniform coverage.

1.2.3.2 Oxford Nanopore Technology

The Oxford Nanopore Technology (ONT) was announced in 2012, but it was made commercially available in 2015 as MinION technology. ONT uses a flow cell to allow massive parallel sequencing. Their flow cell is made of an electrical resistant membrane that contains thousands of tiny pores, each with a diameter of one nanometer. The sequencing principle is based on applying a steady current to the nanopores. The magnitude of current depends on the size and shape of the nanopore. During sequencing, a ssDNA is allowed to pass through the nanopore like a needle and thread so that each nucleotide passes at a time. When a nucleotide passes the opening of the pore, the voltage magnitude is changed. We can use known DNA sequence to capture the unique electrical change for each nucleotide and then allow the DNA molecule that we need to sequence to go through the nanopore opening. The order of the nucleotides on the DNA sequence is determined by matching the patterns. ONT has a higher error rate because the DNA fragments tend to pass faster through the nanopores that may generate errors.

1.3 SEQUENCING DEPTH AND READ QUALITY

1.3.1 Sequencing Depth

The biological results and interpretation of sequencing data for the different sequencing applications are greatly affected by the number of sequenced reads that cover the genomic regions. Usually, multiple sequences overlap over certain regions of the genome. The sequencing depth measures the average read abundance and it is calculated as the number of bases of all sequenced short reads that match a genome divided by the length of that genome if the genome size is known. If the reads are equal in length, the sequencing depth is calculated as

$$\text{Coverage} = \frac{\text{read length (bp)} \times \text{number of reads}}{\text{genome size (bp)}} \quad (1.1)$$

If the reads are not equal in length, the coverage is calculated as

$$\text{Coverage} = \frac{\sum_{i=1}^n \text{length of read } i}{\text{genome size (bp)}} \quad (1.2)$$

where n is the number of sequenced reads.

The sequencing coverage is expressed as the number of times the genome (e.g., 1X, 2X, 20X, ..., etc.).

The sequencing depth affects the genomic assembly completeness, accuracy of de novo assembly and reference-guided assembly, number of detected genes, gene expression levels in RNA-Seq, variant calling, genotyping in the whole genome sequencing, microbial identification and diversity analysis in metagenomics, and identification of protein-DNA interaction in epigenetics. Therefore, it is important to investigate sequencing depth before sequence analysis. The higher the number of times that bases are sequenced, the better the quality of the data.

1.3.2 Base Call Quality

We have already discussed the different sequencing technologies which have different sequencing approaches. However, at the end, each of these technologies attempts to infer the order of the nucleic acid studied. The process of inferring the base (A, C, G, or T) at specific position of the sequenced DNA fragment during the sequencing process is called *base calling*. The sequencing platforms are not perfect and errors may occur during the sequencing process when the machine tries to infer a base from each measured signal. For all platforms, the strength of the signals and other characteristic features are measured and interpreted by the base caller software. Errors affect the sequence data directly and make them less reliable. Therefore, it is critical to know the probability of such errors so that users can know the quality of their sequence data and can figure out how to deal with those quality errors. Most platforms are equipped with base calling programs that assign

a Phred quality score to measure the accuracy of each base called. The Phred quality score (Q-score) transforms the probability of calling a base wrongly into an integer score that is easy to interpret. The Phred score is defined as

$$p = 10^{-Q/10} \quad (1.3)$$

$$Q = -10 \log_{10}(p) \quad (1.4)$$

where p is the probability of the base call being wrong as estimated by the caller software.

The Phred quality scores are encoded using ASCII single characters. All ASCII characters have a decimal number associated with them. However, since the first 32 ASCII characters are non-printable and the integer 33, which is the decimal number for the exclamation mark ASCII character “!”, the Q=0 is the exclamation mark and the encoding that begins with “!” as zero is called Phred+33 encoding. Illumina 1.8 and later versions use this Phred+33 encoding (Q33) to encode the base call quality in FASTQ files. The older Illumina versions (e.g., Solexa) used Phred+64 encoding, in which the character “@”, whose decimal number is 64, corresponds to Q=0. Table 1.1 shows the Phred quality score (Q), corresponding probability (P), and the decimal number and ASCII code. For instance, when the probability of calling a base is 0.1, the Phred score will be 10 (Q=10), but instead of giving the number 10, that quality score is encoded as the plus sign “+”.

Higher Q scores indicate a smaller probability of error and lower Q scores indicate low qualities of the base called which is more likely that the base was called wrongly. For instance, a quality score of 20 indicates the chance of making an error rate (1 error) in 100, corresponding to 99% call accuracy. In general, the Q-score of 30 is considered a benchmark

TABLE 1.1 Phred Quality Score and ASCII_BASE 33 (Q33)

Q	P	ASCII		Q	P	ASCII		Q	P	ASCII	
0	1.00000	33	!	15	0.03162	48	0	30	0.00100	63	?
1	0.79433	34	“	16	0.02512	49	1	31	0.00079	64	@
2	0.63096	35	#	17	0.01995	50	2	32	0.00063	65	A
3	0.50119	36	\$	18	0.01585	51	3	33	0.00050	66	B
4	0.39811	37	%	19	0.01259	52	4	34	0.00040	67	C
5	0.31623	38	&	20	0.01000	53	5	35	0.00032	68	D
6	0.25119	39	‘	21	0.00794	54	6	36	0.00025	69	E
7	0.19953	40	(22	0.00631	55	7	37	0.00020	70	F
8	0.15849	41)	23	0.00501	56	8	38	0.00016	71	G
9	0.12589	42	*	24	0.00398	57	9	39	0.00013	72	H
10	0.10000	43	+	25	0.00316	58	:	40	0.00010	73	I
11	0.07943	44	,	26	0.00251	59	;	41	0.00008	74	J
12	0.06310	45	-	27	0.00200	60	<	42	0.00006	75	K
13	0.05012	46	.	28	0.00158	61	=	43	0.00005	76	L
14	0.03981	47	/	29	0.00126	62	>	44	0.00004	77	M

TABLE 1.2 Phred Quality Score and Error Probability and Base Call Accuracy

Q	Error Probability	Base Call Accuracy (%)	Interpretation
10	0.1	90	1 error in 10 calls
20	0.01	99	1 error in 100 calls
30	0.001	99.9	1 error in 1,000 calls
40	0.0001	99.99	1 error in 10,000 calls
50	0.00001	99.999	1 error in 100,000 calls
60	0.000001	99.9999	1 error in 1,000,000 calls

for good quality in the high-throughput sequencing (HTS). Table 1.2 shows some of the Q scores and corresponding error probability, base call accuracy, and interpretation.

1.4 FASTQ FILES

The sequencing technologies like Illumina are provided with the Real-Time Analysis (RTA) software that stores individual base call data in intermediate files called BCL files. When the sequencing run completes, these BCL files are filtered, demultiplexed if the samples are multiplexed, and then converted into a sequence file format called FASTQ. There will be a single FASTQ file for each sample for a single-end run and two FASTQ files (R1 and R2) for each sample for a paired-end run: R1 file for forward reads and R2 for reverse reads. The FASTQ files are usually compressed and they may have the file extension “*.fastq.gz”.

A FASTQ [7] file is a human-readable file format that has become de facto standard for storing the output of most HTS technologies. A FASTQ file consists of a number of records, with each record having four lines of data as shown in Figure 1.6.

The first line of each record of a FASTQ file begins with the “@” symbol and this line is called the read identifier since it identifies the sequence (read). A typical FASTQ identifier line of the reads generated by an illumine instrument looks as follows:

```
@<instrument>:<run num>:<flowcell ID>:<lane>:<tile>:<x>:<y>:<UMI>
<read>:<filtered>:<control num>:<index>
```

Table 1.3 describes the elements of the Illumina FASTQ identifier line and Figure 1.6 shows an example FASTQ file with three read records. The sequence observed in the index sequence (part of the adaptor) is written to the FASTQ header in place of the sample number. This information can be useful for troubleshooting and demultiplexing. However, these metadata elements may be altered or replaced by other elements especially when they are submitted to a database or altered by users.

The second line of the FASTQ file contains the bases inferred by the sequencer. The bases include A, C, G, and T for Adenine, Cytosine, Guanine, and Thymine, respectively. The character N may be included if the base in a position is ambiguous (was not determined due to a sequencing fault).

The third line starts with a plus sign “+”, and it may contain other additional metadata or the same identifier line elements.

TABLE 1.3 Illumina FASTQ Identifier Line Elements [8]

Identifier Line Element	Description
@	The beginning of the read identifier line
<instrument>	Instrument ID or sequence ID
<run num>	The number of the run on the instrument
<flowcell ID>	The flow cell ID
<lane>	The number of lane where the read was sequenced
<tile>	The number of the tile where the read was sequenced
<x>	The X-coordinate of the DNA cluster
<y>	The Y-coordinate of the DNA cluster
<UMI>	Only if a unique molecular identifier (UMI) is used
<read>	The read number (1 for single read or 2 for paired end)
<filtered>	Y if the read passed the filter and N if didn't pass
<control num>	0 (none of the control bits are on) or an even number
<index>	The sample number or read index

```

@M00573:58:000000000-KH8JC:1:1101:17472:1855 1:N:0:TCTAGCAAC+GACCAGAGCA
TCTTTCGCTAGCATTTAGTGTGCCCCAGCTTTAGTAGGTAAACACAGCTAAACACATCACCACATAATATGGAGCATTTCTTTAAGAAAGT
+
1->AAD1>A11@1FGGF33F33GFDE1FG00AAGHHHHCBGBA1D2D22FE1F/0A00FGB221F/FEEFGHHB21FB0E/FGBGGAEHBGF2FB1D0FAGFHHHHHEBDBB0EG
@M00573:58:000000000-KH8JC:1:1101:13078:1900 1:N:0:TCTAGCAAC+GACCAGAGCA
CAACTGGACCAACTGTTCCCATATGTCATTGACATGTCACACTGGCTGTGAGGTTAATGTTGCTACTGTTGAAACACCTTAATAGTCCTCACTTCTCAAAGAAAAGGT
+
11->A111@B111@FBGGFGFBG3GFHHBHHF1FFHHHFFFFB0AGHFEG?ECC1EEGHFFGGHHHFHHF2FGFHHEHF2F1FEHHGEGBFHHFHGHHHHHHFFBD000FB0EGG
@M00573:58:000000000-KH8JC:1:1101:18622:1921 1:N:0:TCTAGCAAC+GACCAGAGCA
CAACATGCTAGAAAACCTACTGAGATATTGAGTGTGGATAAGCCAGTAATTCTAACATAGTCCTGGCACTAGTGTAGGTGCACTTAATGGCATTACTGTATGTGATGTC
+
11->1CFDFFB111A1FGFFGDHFBCFGHHH3FHCHFEA?EFGEB2EG0AGHBHHHHHFGFHGGHGFHHBHE?ECGHFFHEHHBFHFGFHHGHFBEGHHFHHHHHHFB2FGHH

```

FIGURE 1.6 A FASTQ file format showing three records.

The fourth line of the FASTQ file contains the ASCII-coded string that represents the per base Phred quality scores. The numeric value of each ASCII character corresponds to the quality score of a base in the sequence line.

Researchers usually acquire raw sequencing data for their own research from a sequencing instrument. Raw sequencing data can also be downloaded from a database, where scientists and research institutions deposit their raw data and make it available for public. In either case, the raw sequencing data is usually obtained in FASTQ files. The NCBI SRA database is one of the largest databases of raw data for hundreds of species. The FASTQ files are stored in Sequence Read Archive (SRA) format, and they can be downloaded and extracted using SRA-toolkit [9], which is a collection of programs developed by the NCBI and can be downloaded and installed by the instructions available at “<https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi>”.

For the purpose of demonstration, we will download raw data from the NCBI SRA database. We will use a single-end FASTQ file with the run ID “SRR030834”, whose size is 3.5G. The FASTQ file contains reads sequenced from an ancient hair tuft of 4000-year-old male individual from an extinct Saqqaq Palaeo-Eskimo, excavated directly from culturally deposited permafrozen sediments at Qeqertasussuk, Greenland. To keep file organized, you can create the directory “fastqs” and then download the FASTQ file using “fasterq-dump”

command (make sure that you have installed the SRA-toolkit on your computer and it is on the path):

```
mkdir fastqs
cd fastqs
mkdir single
cd single
fasterq-dump --verbose SRR030834
```

As shown in Figure 1.7, the FASTQ file “SRR030834.fastq” has been downloaded to the directory, and we will use that file to show how to use some Linux commands to perform some operations with that file.

FASTQ files may contain up to millions of entries, and their sizes can be several megabytes or gigabytes, which often make them too large to open in a normal text editor. In general, no need to open a FASTQ file unless it is necessary for troubleshooting or out of curiosity. To display a large FASTQ file, we can use some Unix or Linux commands such as “less” or “more” to display very large text file page by page or “cat” to display the content of the file.

```
less SRR030834.fastq
more SRR030834.fastq
cat SRR030834.fastq
```

If a FASTQ file name ends with the “.gz” extension, that means the file is compressed with “gzip” program. In this case, instead of “less”, “more”, and “cat” commands, use “zless”, “zmore”, and “zcat” commands, respectively, without decompressing the files.

We can also use “head” and “tail” to display the first lines and last lines, respectively. The following command will display the first 15 lines of the file:

```
head -15 SRR030834.fastq
```

If a FASTQ file is large, we can compress it with the “gzip” program to reduce its size more than three times. Compressing the “SRR030834.fastq” file with gzip will reduce its size to less than one gigabyte.

```
gzip SRR030834.fastq
```

The file name will become “SRR030834.fastq.gz”.

```
spots read      : 14,905,497
reads read     : 14,905,497
reads written  : 14,905,497
(base) hamid@node2:~/junk/fastqs/single$
```

FIGURE 1.7 Downloading a FASTQ file from the NCBI SRA database.

Use “gzip -d” to decompress a compressed file.

```
gzip -d SRR030834.fastq.gz
```

If you need to know the number of records in a FASTQ file, you can use a combination of “cat” or “zcat” and “wc -l”, which counts the number of lines in a text file. Remember that a record in a FASTQ file has 4 lines. We can use the Unix pipe symbol “|” to transfer the output of the “cat” command to the “wc -l” command. The following command line will count the number of records stored in the FASTQ files:

```
cat SRR030834.fastq | echo $((`wc -l`/4))
```

If we need to display the file name and read count for multiple files, with the “.fastq” file name extension, in a directory, we can use the following script:

```
for filename in *.fastq;
do
echo -e "$filename\t`cat $filename | wc -l | awk '{print $1 / 4}'`"
done
```

To display a FASTQ file in a tabular format, you can use the “cat” command and then use the Unix pipe to transfer the output to the “paste” command, which converts the four lines of the FASTQ records into tabular format.

```
cat SRR030834.fastq | paste - - - ->SRR030834_tab.txt
```

The command will store the new tabular file in a new file “SRR030834_tab.txt”. You can open this file in any spreadsheet, or you can display it as follows:

```
less -S SRR030834_tab.txt
```

Creating a tabular file from a FASTQ file will help us to perform several operations such as sorting of the entries, filtering out the duplicate reads, extracting read IDs, sequences, or quality scores, and creating a FASTA file. We expect that the format of the identifier lines of a FASTQ file is consistent. If you display “SRR030834_tab.txt”, you will notice that some of the identifier line fields are separated by spaces, and if we consider the space as a column separator, the IDs will be in the first column and the sequence will be in the fourth column. However, this column order may be different in tabular files extracted from other FASTQ files. Assume that we wish to extract only the IDs and sequences from “SRR030834_tab.txt” in a separate text file, then we can use the “awk” command as follows:

```
awk '{print $1 "\t" $4}' SRR030834_tab.txt>SRR030834_seq.txt
```

The “awk” command extracts the first column and fourth column from “SRR030834_tab.txt” and prints the two columns separated by a tab. The output is directed to a new text file “SRR030834_seq.txt” (Figure 1.8).

Linux commands allow us to do multi-step operations. Assume that we want to create a FASTA file from the FASTQ file; we can do that in multiple steps. First, we need to extract both IDs and sequences in a file as we did above, then we can remove “@” symbol leaving only the IDs, then we need to add “>” in the beginning of each line with no space between the “>” and the IDs, and finally, we separate the two columns, forming the definition line (defline) of FASTA and the sequence, store them in a file, and delete the temporary files.

```
cat SRR030834.fastq | paste - - - \
>SRR030834_tab.tmp
awk '{print $1 "\t" $4}' SRR030834_tab.tmp \
| sed 's/@//g'>SRR030834_seq.tmp
sed -i 's/^/>/' SRR030834_seq.tmp
awk '{print $1, "\n" $2}' SRR030834_seq.tmp \
>SRR030834.fasta
rm *.tmp
```

In the FASTA format, as shown in Figure 1.9, each entry contains a definition line and a sequence. The defline begins with “>” and can contain an identifier immediately after “>” (no whitespace in between).

```
@SRR030834.64 CTCTGGGAGGCTGAGGGGGCGGATCCCGAAGTCAGGGGATCGAGGCCATCATGGGAAACAGAGTNANAN
@SRR030834.65 CTCTACTAAAAAATACAAAAAATTAGCCAGGCCTGGCAGGAGATCGGAAGAGCACACGCTCTGAANTN
@SRR030834.66 GAACGGGCTCTGGGCAAGGGCAGCATTAGCTGGCATCTTAAGATCGGAAGAGCACACGCTGANCN
@SRR030834.67 TCCTGGGGACTCCAGGCCCCCAGAGACAGAGCCTCCGAGGCACAGATCGGAAGAGCACACGCTGNN
@SRR030834.68 CAGGGTGTGGATTACAGGCCAGGGCTGAGGCCACCCGGGGGAAACCTAGATCGGAAGAGCACACGCTGN
@SRR030834.69 AAATACAAAAAAATTAGCCCCGGCTGGCGGGGGCTGTAGTCCCAGAGATCGGAAGAGCACACGTCN
@SRR030834.70 GTATTAGGAACCCCTGCCCTGGAGGATGCTTCGGCCCGCTAATGGAAGATCGTAATCGCACACGCTTN
@SRR030834.71 GAGTTGAGACCAAGCCGCTGCCAATATGGTAAACCCATCTACTAAAGATCGGAAGAGCACACGCTTN
@SRR030834.72 TAAAGAAAAGATGGCAATCGGCCAGGGCGGTGGCTCCCGCTGTAGTCGGCATCGGAAGAGCACACCTN
@SRR030834.73 GTCTGTGGGTTAGAACGCAAGGCCAGGTCCCCCTGGACACGCCGTGTGGAGGGGTTGGGGAGTGGAGATN
@SRR030834.74 ATCTGCCAAGGCCGGCGTCCAGGGTGGGGCAGATCGGAAGAGCACACGCTTAACTCCAGTCACACAN
@SRR030834.75 CGCCCTCACCTCCCGAACGGGGCGGTGGCCGGGGGGGGCGACATCCCAGATCGGAAGAGCACAN
@SRR030834.76 TCCCTCTAACATGTCTTCAATATGACACTACAAACCACCTGATGCTTCCATGTGGCCCTTAAN
@SRR030834.77 AATCTACAAAGAACTCAAATTTCCTGGCTTACAGCCAGGAAGAAAAGATCGGAAGAGCACACGTT
```

FIGURE 1.8 Extracting IDs and sequence of a FASTQ file.

```
>SRR030834.127
AAATTCTAACCATTTCTTCACTTACATGTTCTGTGAGATCGGAAGAGCACACGCTCTGAACCTCC
>SRR030834.128
TTGCACTGAGCCAAGATTGCCACTGCACTCCAGCCTTGGCAGAGCGAAACTAGATCGGAAGAGCA
>SRR030834.129
TCAATAGAACATGCCAATGTATCATCTGAGTATGAAAAGATGTACAATAATAGCATTCTATTGAAGATCG
>SRR030834.130
CAATCGAACATGGAATCGAACGGTAGGAATGGAGTGGAAAGATCGGAAGAGCACACGCTCTGAACCTC
>SRR030834.131
GCAGCCGCCAGGCCAGCGCCGCCCAAGATCGGAGGAGGACACGTATGAACCTCCGTACACAGTGA
>SRR030834.132
GTCCCCAACATGAGCCTCAGCCGGAGTAACCGTGGCGGAAGATCGGAAGAGCACACGCTCTGAACCTCA
```

FIGURE 1.9 Extracting FASTA sequence from the FASTQ file.

1.5 FASTQ READ QUALITY ASSESSMENT

The quality control for obtaining good sequencing data begins by isolation of pure nucleic acid from the samples of interest. After isolation, the nucleic acid quality is usually assessed before sequencing. The nucleic acid must be pure and have sufficient concentration as recommended by the kits used for library preparation. If the Nanodrop is used, the purity of the nucleic acid is measured by the ratio of light absorbances at 260 and 280 nm. A ratio of ~1.8 is generally accepted as pure for DNA and a ratio of ~2.0 is generally accepted as pure for RNA. Similarly, absorbance at 230 nm is usually due to other contamination. For pure nucleic acid, the 260/230 value is often higher than the respective 260/280 value. The purity of the RNA is better to be measured by the Bioanalyzer which provides the RNA integrity (RIN). The RIN ranges from 1 to 10, where 10 is the best RNA integrity or non-degrading RNA. Most commercially available RNA library preparation kits require an RNA of a RIN value greater than 7 (RIN>7) for proper library construction.

Another quality control check point is performed after library preparation and before sequencing. The quality of the nucleic acid libraries can also be assessed to ensure that there are no remaining adaptor primers that may cause adaptor dimers in the sequenced DNA fragments. The adaptor dimers are DNA produced from complete adaptor sequences that can bind and cluster on the flow cell and generate contaminating reads, which negatively impact the quality of sequencing data.

The quality assurances in the steps of nucleic acid isolation and removing of adaptor primers after library preparation help in producing high-quality reads. However, errors can also be generated during the sequencing. In Illumina instruments, errors in base call may be made due to failure to terminate the synthesis (polymerase remains attached), forming leading strands that are longer than normal, or due to failure to reverse synthesis termination in the washing cycle, forming lagging strands. The base call software converts fluorescence signals into actual sequence data with quality scores, which will be stored in FASTQ files.

A post-sequencing quality check must be carried out to assess the read quality to ensure that the sequence data looks good and there are no problems or biases that lead to inaccurate or misleading results. FastQC [10] is the most popular program for assessing the quality of the sequencing data produced by high-throughput instruments. FastQC provides a simple way to assess the quality of the raw data and generates summary and simple graphical reports that we can use to have a clear idea about the overall quality of the raw data and any potential quality problem. FastQC can be downloaded for all platforms from "<https://www.bioinformatics.babraham.ac.uk/projects/fastqc/>". You can use the following steps to install the latest FastQC version, as of this time, on Linux (Ubuntu):

Use "wget" command to download the current version; the current version is v0.11.9, which may change in the feature.

```
 wget https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
 fastqc_v0.11.9.zip
```

Decompress the zipped file with unzip:

```
unzip fastqc_v0.11.9.zip
```

A directory named “FastQC” containing the program files will be created.

Change to that directory with:

```
cd FastQC
```

This directory contains several files but we will focus only on “fastqc” file, which is the program command that we will use and “Configuration” directory which contains three files “adapter_list.txt”, “contaminant_list.txt”, and “limits.txt”, which can be used to configure the QC report.

Use the Linux command “chmod” to make “fastqc” file executable:

```
chmod 755 fastqc
```

You may need to add the program to the Linux path so that you can use it from any directory. To do that, open the file “`~/.bashrc`” with any text editor and add the program path to the end of the file, save the file and exit.

```
export PATH="/home/mypath/FastQC":$PATH
```

You may need to replace “`/home/mypath`” in the above statement with the path of your downloaded FastQC.

You need to restart the terminal or you can use “source” command to make that change active.

```
source ~/.bashrc
```

Finally, test the FastQC from any directory with the following:

```
fastqc -h
```

If the path was set correctly, you will see the program help screen.

On Linux, the FastQC program can also be installed simply by running “`sudo apt-get install fastqc`”. Once it has been installed, you do not need any further configuration as we did above. Although any one of the two installation methods works, in the first method, you will know where you download the program and you can reach the “Configuration” directory easily to make changes to any of the three configuration files stored there.

The FastQC program can be run in one of two modes: either as (i) an interactive graphical user interface (GUI) in which we can load FASTQ files and view their results or (ii) as a non-interactive mode on a command line where we can specify the FASTQ files and then it will generate an HTML report for each file. However, if we run it non-interactively, to display the report, we will need to open the HTML files on an Internet browser. The command-line mode allows FastQC to be run as part of an analysis pipeline. FastQC uses a FASTQ file as

an input and generates quality assessment reports including per base sequence quality, per tile sequence quality, per sequence quality scores, per base sequence content, per sequence GC content, per base N content, sequence length distribution, sequence duplication levels, overrepresented sequences, adaptor content, and k-mer content. FastQC supports all variants of FASTQ formats and gzip-compressed FASTQ files.

We will download some public single-end FASTQ files from an NCBI BioProject with an accession “PRJNA176149” for practicing purpose. The SRA files of this project contain genomic single-end reads of *Escherichia coli* str. K-12. To keep the files organized, we can create the directory “ecoli” using “mkdir ecoli” and then move it inside this directory “cd ecoli” and save the following IDs (each in a line) in a text file with the file name “ids.txt” using any text editor:

```
SRR653520
SRR653521
SRR576933
SRR576934
SRR576935
SRR576936
SRR576937
SRR576938
```

Then, run the following script to create the subdirectory “fastQC” and to download the FASTQ files associated with the IDs stored in the “ids.txt” file into the directory:

```
mkdir fastQC
while read f;
do
fasterq-dump \
--outdir fastQC "$f" \
--progress \
--threads 4
done<ids.txt
```

Once the raw FASTQ files have been downloaded, we can use the command “ls -lh fastQC” to display the file names as shown in Figure 1.10.

```
(base) hamid@node2:~/junk/fastqs/fastqc$ ls -lh
total 12G
-rw-rw-r-- 1 hamid hamid 693M Sep  5 12:17 SRR576933.fastq
-rw-rw-r-- 1 hamid hamid 1.9G Sep  5 12:19 SRR576934.fastq
-rw-rw-r-- 1 hamid hamid 1.5G Sep  5 12:20 SRR576935.fastq
-rw-rw-r-- 1 hamid hamid 2.3G Sep  5 12:21 SRR576936.fastq
-rw-rw-r-- 1 hamid hamid 1.1G Sep  5 12:22 SRR576937.fastq
-rw-rw-r-- 1 hamid hamid 1.3G Sep  5 12:23 SRR576938.fastq
-rw-rw-r-- 1 hamid hamid 1.6G Sep  5 12:15 SRR653520.fastq
-rw-rw-r-- 1 hamid hamid 1.5G Sep  5 12:16 SRR653521.fastq
```

FIGURE 1.10 The names of the downloaded FASTQ files.

On Linux terminal, we can use FastQC non-interactively and later we will display the generated reports on an Internet browser. But before running FastQC, it is important to know about the “limits.txt” file in the “Configuration” directory. This file contains the default values for the FastQC options, and we can use it to determine which report to generate. Use a text editor of your choice to open that file and study its content. In most cases, no change is needed. At this point, we will change only

```
kmer ignore 1      to      kmer ignore 0
```

Then, save the file and exit. This change is necessary to include k-mer report when we run the program.

The following is a simple syntax for running the FastQC program non-interactively on the command line:

```
fastqc seqfile1 seqfile2 .. seqfileN
```

The input can be a single FASTQ file name or multiple file names separated by whitespaces. The FastQC program has several options that can be displayed using the following command:

```
fastqc --help
```

Since we have downloaded the eight *E. coli* raw FASTQ files above and stored them in the “fastQC” directory, we can either run the program for each file or provide all file names as input as shown in the above syntax. However, the efficient way is to use the bash commands if we are using a Linux/Unix platform. The following bash script creates a directory “qc”, changes to “fastQC” directory where the FASTQ files are stored, stores the file names in a variable “filename”, then runs the FastQC program non-interactively, and finally saves the QC reports in the “qc” directory:

```
mkdir qc
cd fastQC
filenames=$(ls *.fastq)
fastqc $filenames \
--outdir ../qc \
--threads 3
cd ..
```

We can also simply use the following command:

```
mkdir qc
cd fastQC
fastqc *.fastq --outdir ../qc --threads 3
```

The QC reports of the FASTQ files will be stored in the “qc” directory. FastQC will generate an HTML file “*_fastqc.html” and a zipped file “*_fastqc.zip” for each FASTQ file.

The zipped file contains the images and the text files that will be displayed on the HTML when it is opened on an Internet browser. We can display the HTML files from the Linux terminal using Firefox. If Firefox is not installed, use “sudo apt-get install firefox” to install it. If you are using Putty to access a remote computer, you may need to enable “X11 Forwarding”.

Use the following syntax to display the QC reports:

```
firefox HTMLfile1 HTMLfile2 .. HTMLfileN
```

Alternatively, you can use a bash script as follows:

```
cd qc
htmlfiles=$(ls *.html)
firefox $htmlfiles
cd ..
```

The Firefox Internet browser will display the QC FastQC reports of all FASTQ files, each on a separate tab as shown in Figure 1.11. Click a tab to move from a report to another.

A QC FastQC report includes a summary, basic statistics, and the quality control metrics that we will discuss in some detail. Summary, on the left-hand side, provides a quick visual report to the different QC metrics available in the report and allows us to identify any existing problem. With each metric title, Summary displays a sign indicating whether there is any potential quality problem; the green color indicates a normal metric, orange indicates a slightly abnormal metric, and red indicates a quality problem associated with the metric. A metric with an orange warning requires an attention and correction if it is possible and a metric with a red warning indicates that there is a clear fault in the sequence data and it is recommended to fix it before proceeding to the analysis step. Summary also

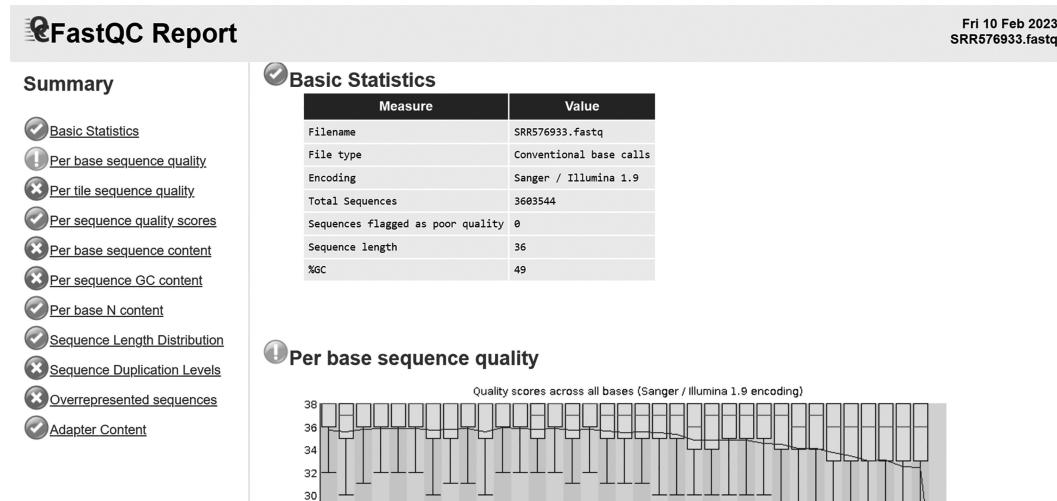


FIGURE 1.11 QC FastQC reports.

acts as a table of content. Clicking a Summary item will take you to that item graph. In the following, we will discuss each item that may be included on the QC report.

1.5.1 Basic Statistics

Basic Statistics table, as shown in Figure 1.11, includes summary information about the FASTQ file analyzed with the FastQC. Basic Statistics title is always green and does not show a warning sign. The information in the basic statistics includes the file name, file type, encoding, total sequences (number of reads), sequence flagged, sequence length, and %GC content. “Filename” field indicates the name of the FASTQ file analyzed. “File type” field indicates whether the analyzed file was generated by the conventional base calling or by another means. “Encoding” field indicates the ASCII encoding for the Phred quality score. Since the encoding in the table is “Sanger / Illumina 1.9”, the Phred+33 encoding (Q33) was used to encode the per base quality scores (as described above). “Total sequences” field shows the number of records in the FASTQ file. “Filtered Sequences” field shows the number of removed reads if the FASTQ file is in Casava format (Casava FASTQ file will be discussed in Chapter 7). “Sequence Length” field shows either the length of the shortest and longest sequence if the length of the reads is variable or a single value if all reads in the FASTQ file have the same length. The “%GC” field shows the overall percentage of GC content of all bases in all reads.

We should pay attention to “Total Sequences” if the analyzed FASTQ files are paired end; the number of sequences must be the same in both files (forward and reverse files) as shown in Figure 1.12. Otherwise, some programs used in the analysis may complain because they expect matched pairs and in the same order in both files.

We should also pay attention to “Sequence length”. A range of numbers as shown in Figure 1.13 indicate that the lengths of the sequences in the file are not equal and they can be any length in the range between the two numbers. Some programs used in the analysis may expect that all reads in a FASTQ file have equal length.



Basic Statistics

Measure	Value
Filename	SRR576933.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	3603544
Sequences flagged as poor quality	0
Sequence length	36
%GC	49

FIGURE 1.12 The basic statistics of the QC report.

Measure	Value	Measure	Value
Filename	SRR10321130_1.fastq	Filename	SRR10321130_2.fastq
File type	Conventional base calls	File type	Conventional base calls
Encoding	Sanger / Illumina 1.9	Encoding	Sanger / Illumina 1.9
Total Sequences	651670	Total Sequences	651670
Sequences flagged as poor quality	0	Sequences flagged as poor quality	0
Sequence length	35-301	Sequence length	35-301
%GC	62	%GC	61

FIGURE 1.13 The basic statistics of the paired-end FASTQ files.

The GC content (%GC) is important for showing sequencing problem due to bias. There is a relationship between GC content and read coverage across a genome. Some short-read sequencers may tend to sequence the region with higher GC content more or less than the region with the lower GC content [11]. The average of the GC content of bacterial genome varies from less than 15% to more than 75% [12], and the average genomic GC content of most eukaryotes lies somewhere within 40%–50% [13]. Very small or very large GC content may indicate a potential sequencing bias problem that we may need to fix.

1.5.2 Per Base Sequence Quality

Per Base Sequence Quality graphs (Figure 1.14) show box plots of the quality distributions on each position across all bases of the reads in the forward and reverse FASTQ files. The position indexes are plotted in the *x*-axis against the Phred quality scores in the *y*-axis. The higher the quality score, the better the base call.

The maximum limit of the *x*-axis indexes depends on the length of the reads if they have the same length; otherwise, it will be the number of bases in the longest read. Each box plot displays the *five-number summary* of the base quality scores in that position. The five-number summary includes the *minimum quality score* (the lower end of lower whisker), *first quartile* (the lower end of the box), *median* (the red line), *third quartile* (the upper end of the box), and *maximum quality score* (upper end of the upper whisker). The yellow box of a box plot represents the *interquartile range* (IQR), which is the middle 50% of the quality scores of the bases in that position. The longer the box, the more the spread of the quality scores. The blue line on the graph represents the mean of the quality scores.

The background of the graph is divided into three regions. The quality scores on the green region ($Q > 28$) are very good, the scores on the orange region ($20 < Q < 28$) are reasonable, and the scores on the red region ($Q < 20$) are poor. In general, for the reads produced by Illumina, the per base qualities degrade toward the end of the reads.

If the lower quartile of the quality scores for any position is less than 20, an orange warning will be displayed in front of “Per Base Sequence Quality” as shown in Figure 1.14. The red warning (failure) is displayed if the lower quartile for any position is less than 5. We can notice that the average quality of the base drops toward the end of the reads and it is greatly deteriorated in the 36th base. To avoid using low-quality sequence data in the

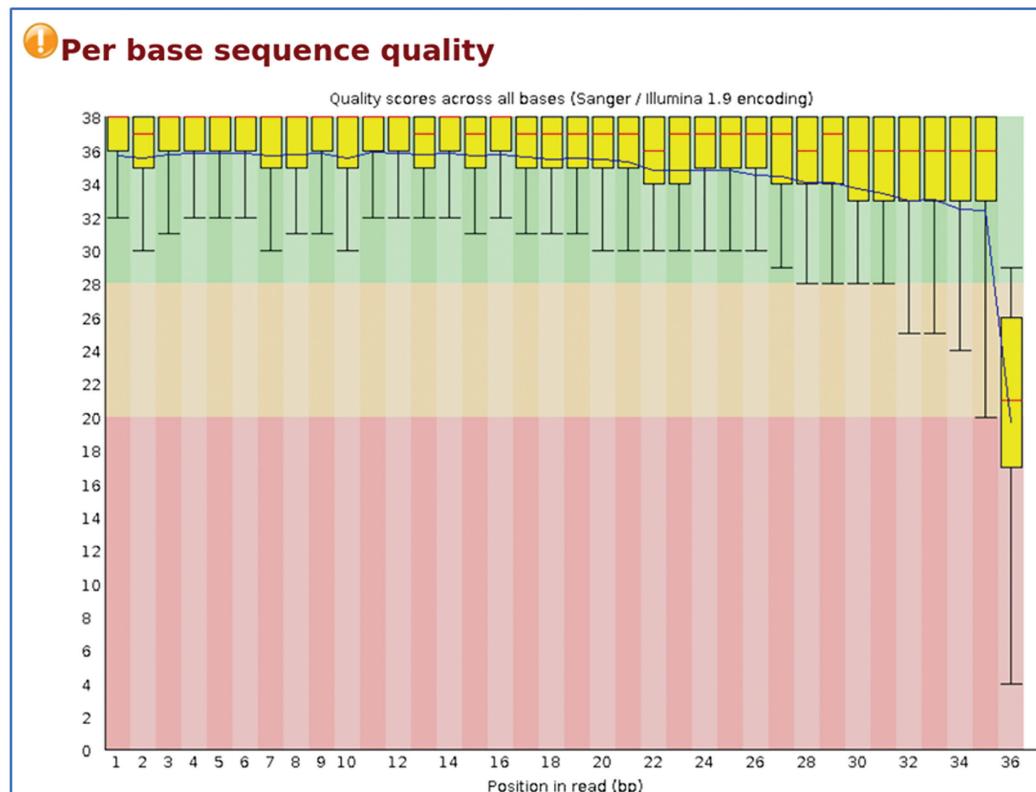


FIGURE 1.14 Per base sequence quality with warning.

analysis, we may need to filter the reads that have low-quality bases or to trim the ends of the reads beginning from the 34th base. Figure 1.15 shows a per base sequence quality graph without warning.

1.5.3 Per Tile Sequence Quality

The per tile sequence quality is represented by a heatmap graph that is available only if an Illumina sequencer was used and the reads in the FASTQ file retain the original identifiers including the IDs of the flow cell tiles on which reads were sequenced. Figure 1.16 shows the first two records of the FASTQ file “SRR576933.fastq”. Notice that the identifier line of each record contains the sequence ID, the flow cell ID, lane number, tile number, *x*-coordinate and *y*-coordinate of the tile, and read length.

In the graph, the base position indexes are plotted in the *x*-axis against the physical positions on the flow cells (tile numbers) in the *y*-axis. The base quality is represented by a color scale from blue (cold) to red (hot). The blue color indicates that the quality of the base from the tile is at or above the average for the base in the run. The red color indicates the quality for the base in that tile is worse than the quality for the same base from the other tiles. The graph provides an easy way to track the average quality scores from each tile across all

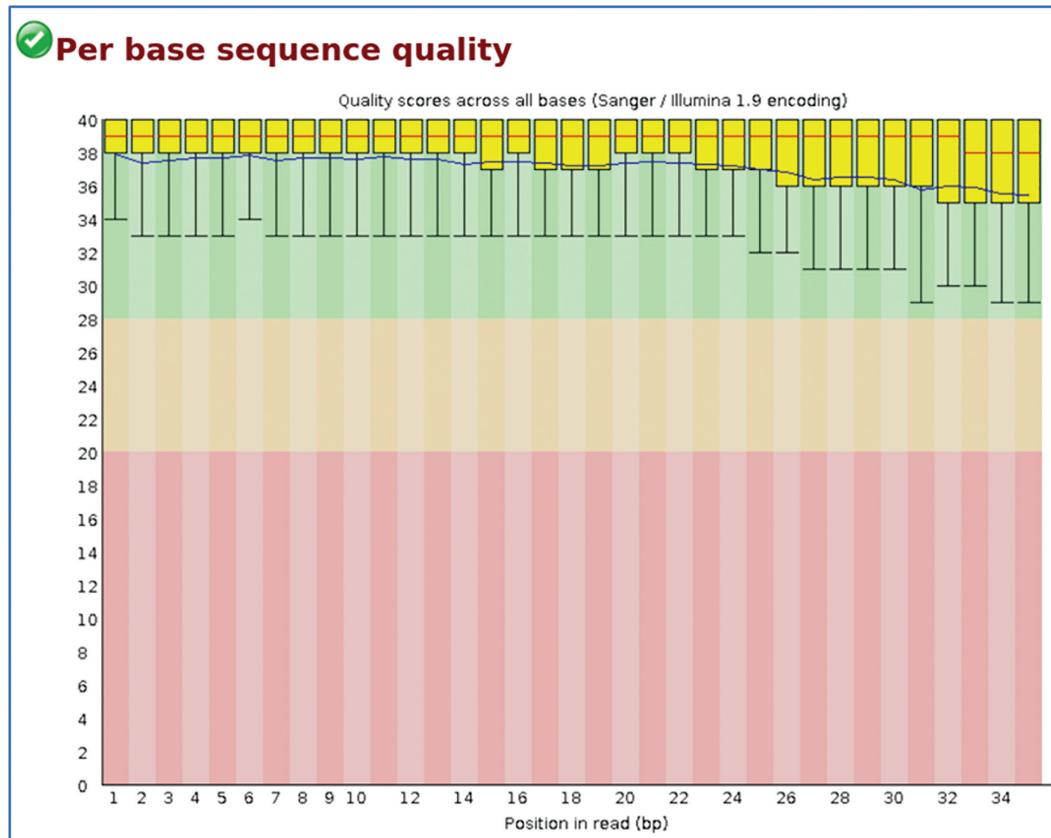


FIGURE 1.15 Good per base sequence quality (with no warning).

```
+SRR576933.3603542 HWUSI-EAS1789_0000:2:96:19837:10688 length=36
C@.=@5AAAA:DCDDEEEE5:=C:DDCBBC?BB5
@SRR576933.3603543 HWUSI-EAS1789_0000:2:96:19838:6574 length=36
GCTAATTTCTTGCAGTTAACAGCAGGAAAACAGTCGAA
+SRR576933.3603543 HWUSI-EAS1789_0000:2:96:19838:6574 length=36
DDD:5C. @?6@=C@CDDDDDCCC-@;C7=A?CA57
@SRR576933.3603544 HWUSI-EAS1789_0000:2:96:19855:18205 length=36
NCGATATCATGGCATGATAATTGGTTTTGCCG
+SRR576933.3603544 HWUSI-EAS1789_0000:2:96:19855:18205 length=36
```

FIGURE 1.16 The last records of the FASTQ file “SRR030834.fastq”.

bases in a base position and to quickly detect if there is any poor quality associated with any tile. A graph with a completely blue color indicates an overall good quality base call from all tiles (Figure 1.17a).

The quality losses associated with the physical positions on the flow cell are of different causes. A random low quality at different positions is usually due to a technical problem with the run such as the flow cell overlapping (Figure 1.17b).

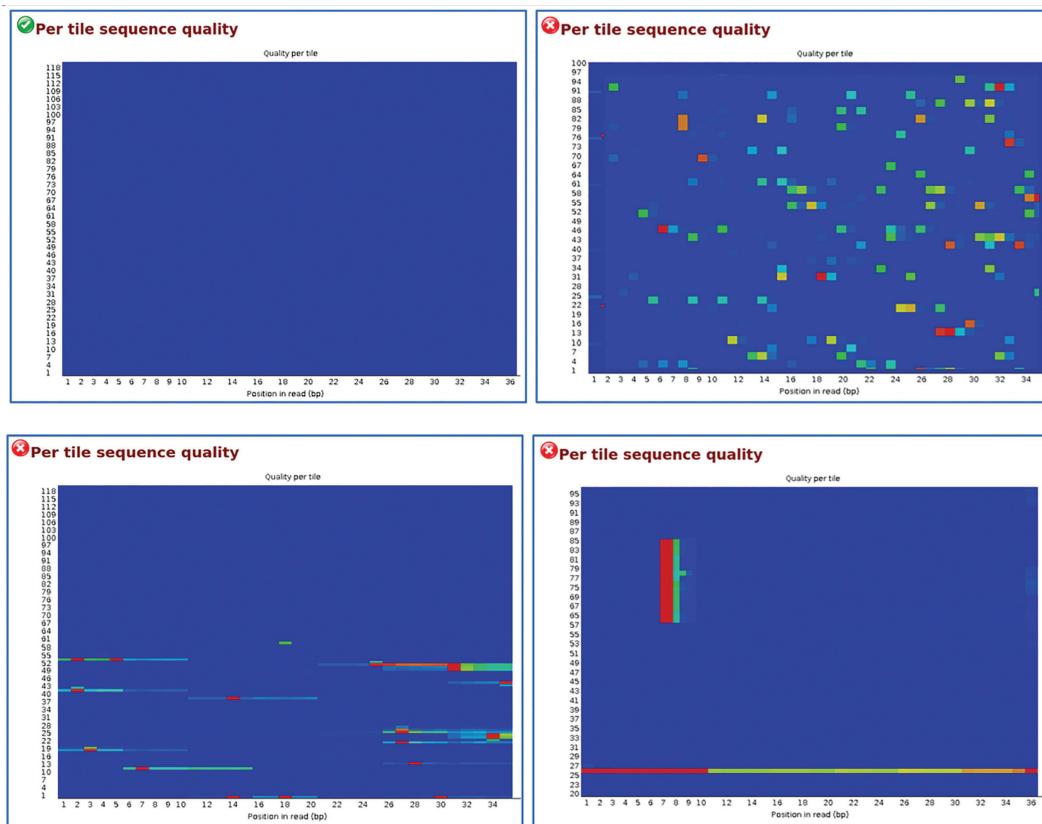


FIGURE 1.17 Per tile sequence quality graphs, some with quality problems.

Reads with such random scattered hot areas (errors) are hard to be fixed. Another cause of the loss of quality is the failure of the imaging system at the edges of the flow cell to read signals (Figure 1.17b). However, the reads in such case can be still useable. The loss of quality that begins from somewhere and continues until the end of the run (Figure 1.17c) is usually due to an obstruction to the imaging system (because of a dirt on the surface of the flow cell). The reads with low quality, in this case, can be filtered out. A final type of errors associated with tiles is a temporary loss of quality in specific base positions (Figure 1.17d) due to an obstruction to few cycles (because of bubbles). The obstruction usually stops the imaging and also prevents the reagents from getting to the DNA template clusters. Such quality problem may introduce false insertion to the reads and it cannot be fixed with sequence trimming since the low-quality bases are not at the end of the reads.

Indeed, not every quality problem associated with tiles can be removed by trimming or filtering out reads originating from the affected tiles. However, the last one may affect several tiles and may extend several reads without lowering the overall sequence score. We should watch out to this kind of fault especially for reads used for variant discovery.

1.5.4 Per Sequence Quality Scores

The per sequence quality score graph is created by plotting the mean sequence quality (Phred scores) in the *x*-axis against read count (frequency) in the *y*-axis. The graph allows us to see if a subset of reads have an overall low quality. The ideal curve is the one that shows the majority of the reads having an overall quality score at or over 30 (Figure 1.18a); a peak is toward the end of the *x*-axis. The presence of a large number of reads with an overall low quality will indicate a systematic problem in the run. A warning sign is displayed if the mean quality score of the majority of reads is below 27 (Figure 1.18b). An error is displayed if the average quality score of the majority of the reads is below 20. The low-quality reads can be filtered out to keep only the reads that pass a quality threshold.

1.5.5 Per Base Sequence Content

The per base sequence content graph depicts the percentage of each of the four bases (A, C, G, and T) called at each position across all reads in a FASTQ file. The positions are plotted in the *x*-axis against the base percentage in the *y*-axis. If there is no bias and library sequencing is random, we will expect no big difference between the distributions of the four bases in each position. The percentage of each base is expected to be close to 25% and the four lines will run approximately parallel to each other as shown in Figure 1.19a. Any deviation from that, such as a bias or a systematic fault, will be suspected, and hence, some sequences may be overrepresented as shown in Figure 1.19b. Higher percentage of some bases at the beginning of the *x*-axis may indicate contaminating remnants of adaptor sequences or other contaminating sequences. A warning is displayed if the difference between any of the four bases is greater than 10% in any position and the failure of this metric occurs if the difference between any four of bases is greater than 20% in any position.

1.5.6 Per Sequence GC Content

The per sequence GC content graph plots the number of reads in the *y*-axis against the mean GC percentage per read in the *x*-axis (Figure 1.20). It depicts the distribution of GC

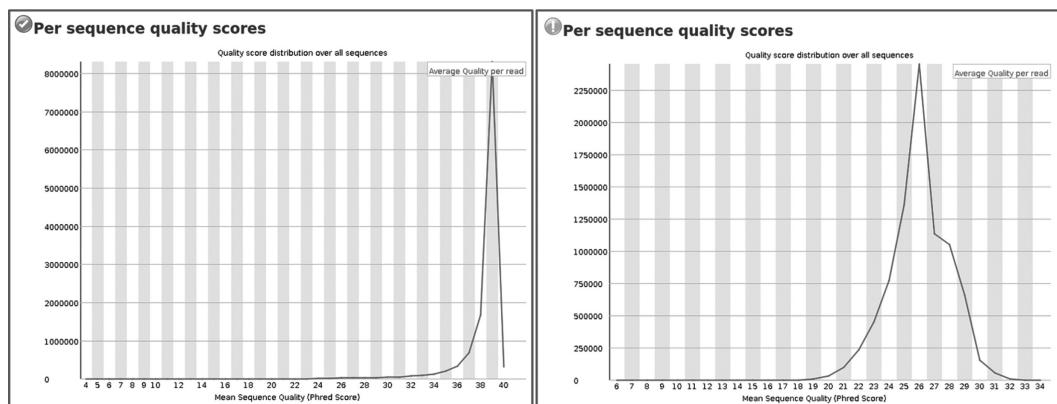


FIGURE 1.18 Per sequence quality scores.

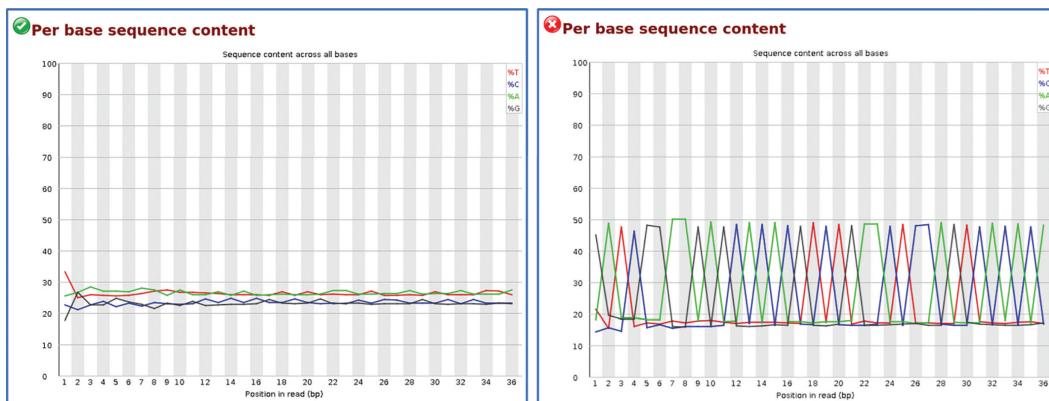


FIGURE 1.19 Normal and abnormal per base sequence content.

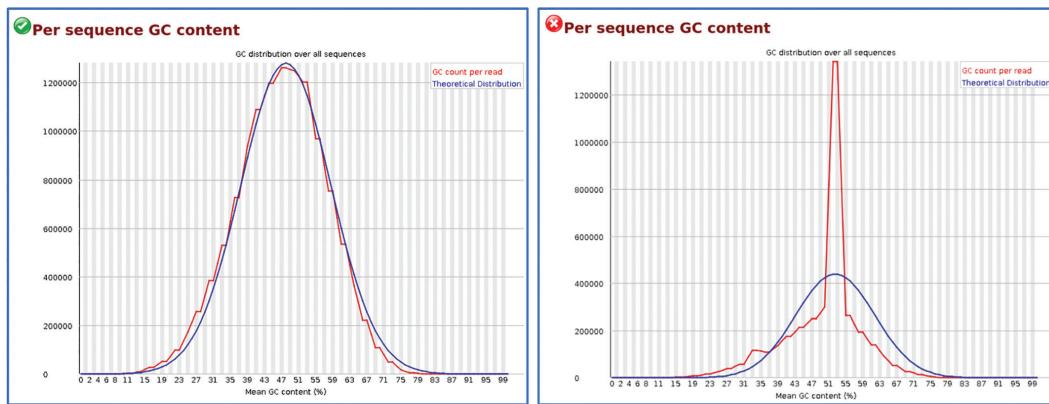


FIGURE 1.20 Normal and abnormal per sequence GC content.

content across the reads in a FASTQ file and then compares it to the theoretical normal distribution of the GC content, which is estimated from the observed data. If there is no sequencing bias and the library is random, we will expect that the observed distribution of the GC content of the reads to be approximately normal and roughly similar to the theoretical distribution in which the central peak corresponds to the overall GC content of the underlying genome. Deviation of the distribution of the per sequence GC content from the normal distribution is an indication of a contaminated library or a fault in the sequencing process. However, a bell-shaped normal curve that deviates from theoretical curve may or may not be biased. In this case, there is a chance that the observed distribution may represent the actual distribution of the genome of the organism; therefore, no warning will be issued.

A warning sign is displayed if the observed distribution deviates from normal distribution by a sum of more than 15% of the reads. A failure sign will be displayed if the distribution deviates by a sum of more than 30% of reads.

1.5.7 Per Base N Content

During the sequencing process, a base is called with a high confidence. However, for some fault, the machine may fail to call any base at a specific position. The “N” character is then placed at that position as an indication of call failure. A few call failures are tolerable; however, if the frequency of “N” is high, that may pose a quality problem. The per base N content graph shows the distribution of “N” at each base position. The N percentages are plotted in the *y*-axis against positions in the *x*-axis. A warning is issued if any position shows an N content of greater than 5% and a failure sign if any position shows an N content of greater than 20%. Figure 1.21 shows the per base N content with no problem.

1.5.8 Sequence Length Distribution

In the library preparation step, DNA molecules are cut into equal fragments to generate reads with equal lengths. Most sequencing instruments run quality control to keep the

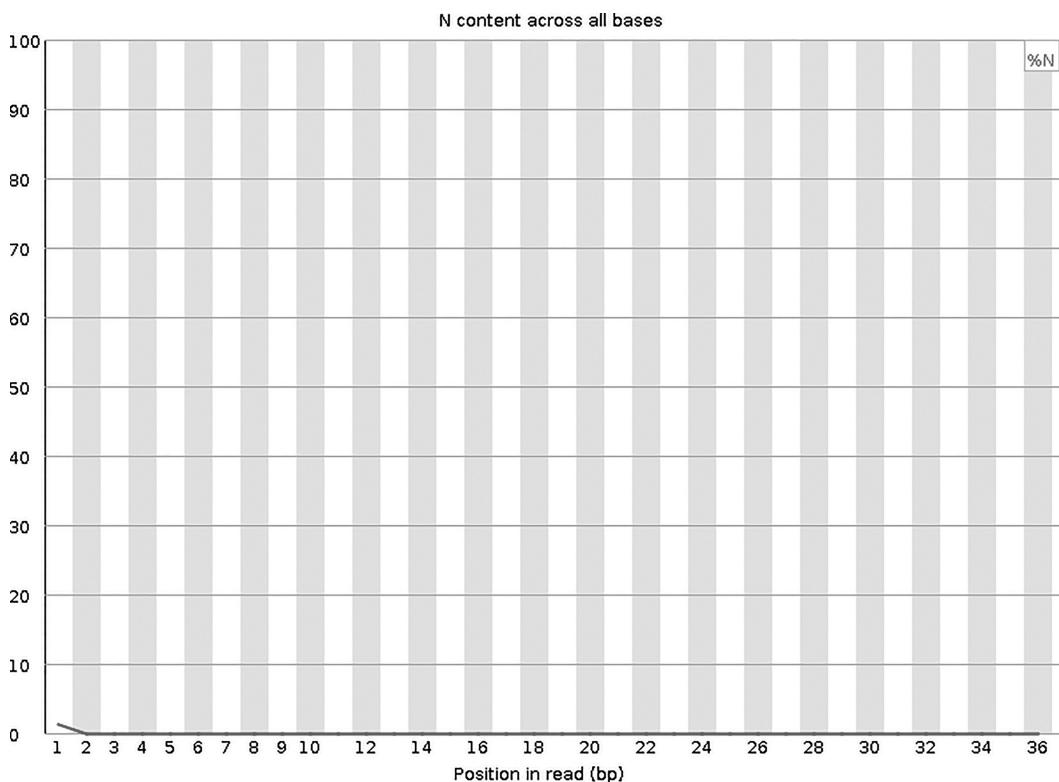


FIGURE 1.21 Per base N content.

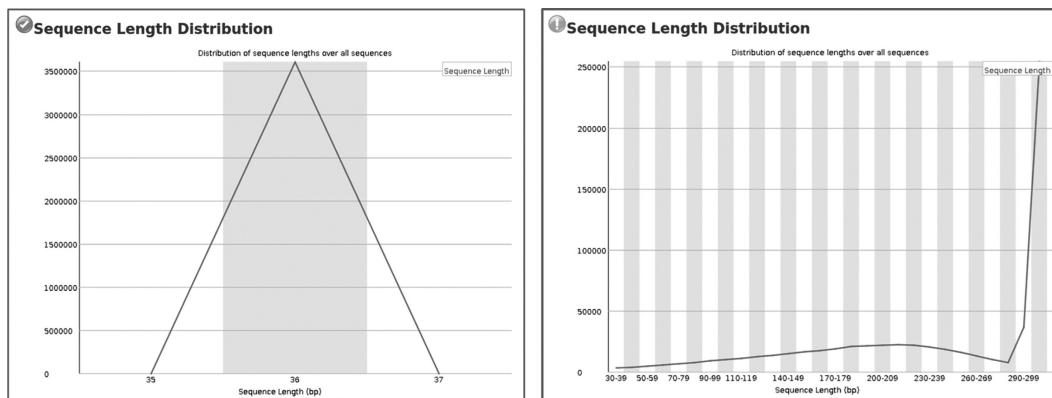


FIGURE 1.22 Sequence length distribution graphs (equal length and variable lengths).

length of the reads equal. However, sometimes reads with unequal lengths are generated especially if the reads are trimmed to remove low-quality bases at the beginning or ends of the reads. The sequence length distribution graph shows the read length distribution.

If the reads are of the same length, the graph will be simple with a single peak at a bar indicating a single value (Figure 1.22a). When reads are of a variable length, the graph will show the relative read count of each read length (Figure 1.22b). A warning is displayed if the reads do not have the same length.

1.5.9 Sequence Duplication Levels

The PCR may be used in sequencing step especially if the concentration of DNA is low, in RNA-Seq and ChIP-Seq for enrichment. The PCR will increase the number of DNA fragments; a single fragment is duplicated several times (exact match). However, well-calibrated sequencing instrument will produce, at the end, a single read for each of the library fragments. Low sequence duplication level may indicate a high level of coverage. In contrast, the high level of duplication indicates a bias due to PCR amplification. The graph of sequence duplication levels plots the percentages of reads against the sequence duplication levels (number of duplicates). Only the first 200,000 reads in a FASTQ file are checked for duplication to save computer memory. The number of duplicates is counted for each read. A big rise may indicate the presence of a large number of reads with high levels of duplication. A warning is displayed if the number of duplicated reads is more than 20% of the total. Figure 1.23a shows that the majority of reads are unique. However, the number of duplicated reads is more than 20% of the total reads; therefore, a warning is issued. Figure 1.23b shows that the number of duplicated reads is more than 50% of the total; therefore, the metric failed.

1.5.10 Overrepresented Sequences

The overrepresented sequences of genomic DNA will indicate a clear bias or contamination due to adaptor dimers. However, in RNA-Seq, the overrepresented sequences can also be

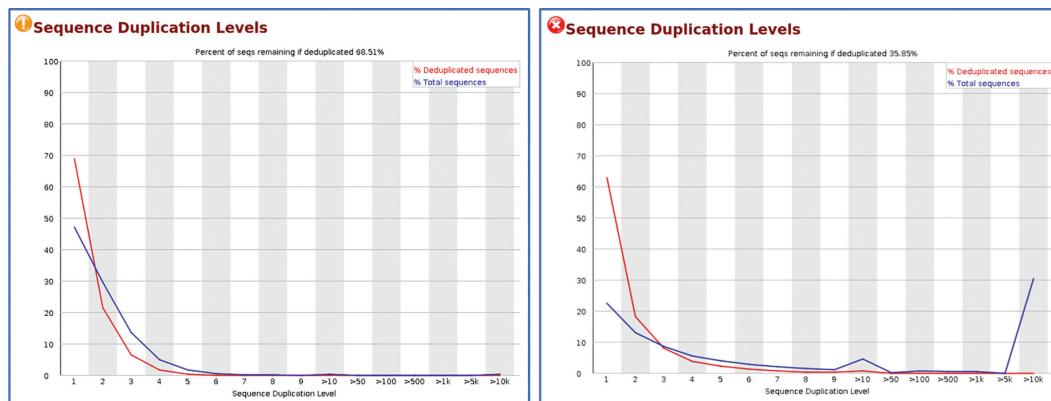


FIGURE 1.23 Sequence duplication levels (warning and failure).

✖ Overrepresented sequences			
Sequence	Count	Percentage	Possible Source
GATCGGAAGAGCACACGTCTGAACCTCCAGTCACACA	1060621	29.432719567181643	TruSeq Adapter, Index 5 (100% over 36bp)
GCTAACAAATACCCGACTAAATCAGTCAGTAAATA	13630	0.37823875606902535	No Hit
NATCGGAAGAGCACACGTCTGAACCTCCAGTCACACA	11728	0.3254573830651159	TruSeq Adapter, Index 5 (97% over 36bp)
GTTAGCTATTTACTGACTGATTTAGTCGGGTATTT	10983	0.304783291115635	No Hit
GATCGGAAGAGCACACGTCTGAACCTCCAGTCACACC	3658	0.10151117899490057	TruSeq Adapter, Index 1 (97% over 36bp)

FIGURE 1.24 Overrepresented sequences.

due to the gene expression and that overrepresentation can be of a biological importance rather than a bias. The overrepresented sequences report is a table that shows the overrepresented sequences, counts, percentage, and possible source. To save memory, only the first 200,000 reads are checked in the FASTQ file; therefore, the list is not exhaustive and other overrepresented sequences may skip the check. For each overrepresented sequence, the FastQC program will search on a database of known contaminants and report the best match that is at least 20 bases in length and has no more than a single mismatch. A warning will be issued if a sequence is overrepresented more than 0.1% of the total and failure will occur if the overrepresentation is more than 1% of the total. As shown in Figure 1.24, five overrepresented sequences are found, three of which are contaminating adaptors and two sequences have no hits. The count and percentage reflect the significance of each of these overrepresented sequences. The count of the first sequence in the table represents 29.4% of the total count of the reads in the FASTQ file. It is clear that this sequence is originated from a primer contamination and it must be removed before analysis.

1.5.11 Adapter Content

The full-length adaptor primers may cause contaminating adaptor dimers of a significant number of reads. The adaptor content graph shows the cumulative percentage count of

Illumina Universal Adapter	AGATCGGAAGAG
Illumina Small RNA 3' Adapter	TGGAATTCTCGG
Illumina Small RNA 5' Adapter	GATCGTCGGACT
Nextera Transposase Sequence	CTGTCTCTTATA
SOLID Small RNA Adapter	CGCCTTGGCCGT

FIGURE 1.25 Some known adaptor sequences.

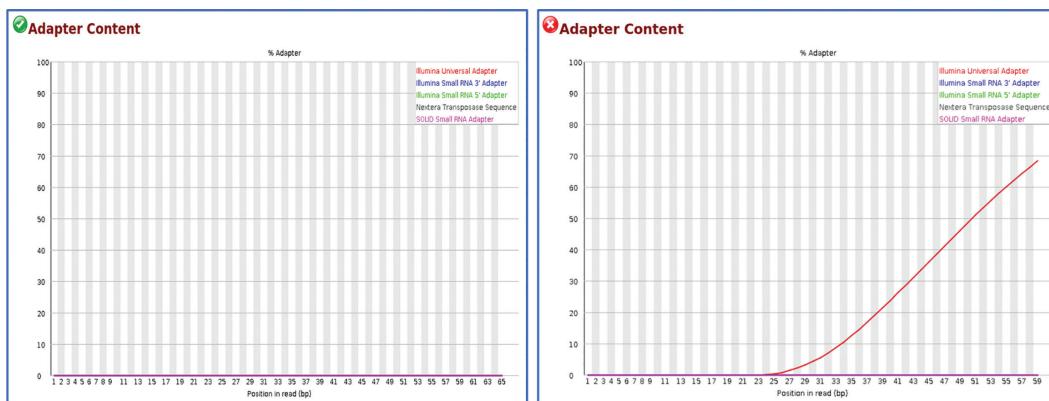


FIGURE 1.26 Adapter content graphs.

the proportion of the reads which contain the adaptor sequences at each position. Known adaptor sequences and description are stored in the “adapter_list.txt” file as shown in Figure 1.25.

K-mers (sequences of k size of bases) are formed from the adaptor sequences in the “adapter_list.txt” file and then the program searches for these k-mers to report the total percentage of the reads which contain these k-mers. The report may discover the sources of bias due to contaminating adaptor dimers in the library.

A warning is raised if any sequence is present in more than 5% of all reads, and a failure occurs if any sequence is present in more than 10% of all reads.

Figure 1.26 shows a FASTQ file with raw reads without adaptor content (left) and a FASTQ file with reads with failed metric due to significant content of Illumina Universal Adaptor.

1.5.12 K-mer Content

The K-mer content graph plots the count of each short nucleotide of length k (default $k=7$) against positions in reads. In a normal k-mer content, k-mers are expected to be represented evenly across the length of the reads. The k-mer content graph shows the positions for the only six most significant k-mers (Figure 1.27). The list of k-mers which are present at specific position with significant abundance will be reported in a table including k-mer sequences, counts, p -values, and expected position. Caution is required when the reads are from RNA-Seq libraries; the significant k-mers may be due to highly expressed gene, and hence, they will have a biological importance.

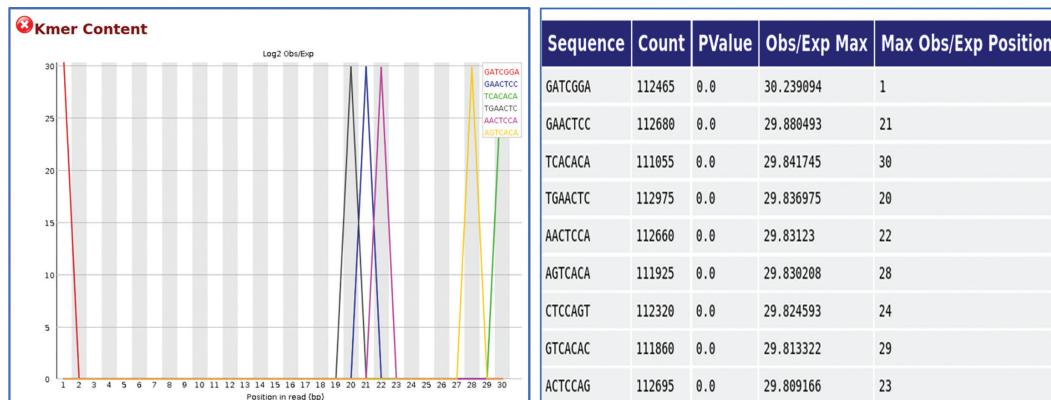


FIGURE 1.27 Failed k-mer content.

1.6 PREPROCESSING OF THE FASTQ READS

In the above, we discussed the assessment of the quality of the reads produced by the HTS instruments to understand the potential errors and biases that may arise from warnings or failures of the quality metrics. Before moving on to the next step for data analysis, errors and biases should be adjusted to avoid incorrect results and misleading interpretation. In general, there are three common approaches to fix the biases resulted from the quality metrics. Those three approaches include (i) trimming the ends of the reads, (ii) removing low-quality reads, and (iii) masking low-quality bases. The use of any of those approaches depends on the quality problem. In the following, we will discuss the most commonly used programs to deal with read quality issues.

The most commonly used software for the processing of raw sequence reads in FASTQ files is FASTX-toolkit [14], which is a collection of command-line programs. The installation instructions of FASTX-toolkit are available at "http://hannonlab.cshl.edu/fastx_toolkit/download.html". We can download and install it on Linux using the following steps:

Create a directory in which you can download the FASTX-toolkit compressed file:

```
mkdir fastxtoolkit
cd fastxtoolkit
```

Download the compressed program file and decompress it:

```
wget http://hannonlab.cshl.edu/fastx_toolkit/fastx_toolkit_0.0.13_
binaries_Linux_2.6_amd64.tar.bz2
tar xvf fastx_toolkit_0.0.13_binaries_Linux_2.6_amd64.tar.bz2
```

Copy the program files from the "bin" directory to "/usr/local/bin" so that it can be executed from any directory on the computer:

```
sudo cp ./bin/* /usr/local/bin
```

fastx_clipping_histogram.pl	fastx_barcode_splitter.pl
fastx_formatter	fastx_clipper
fastx_nucleotide_changer	fastx_collapse
fastq_masker	fastx_nucleotide_distribution_graph.sh
fastq_quality_boxplot_graph.sh	fastx_nucleotide_distribution_line_graph.sh
fastq_quality_converter	fastx_quality_stats
fastq_quality_filter	fastx_renamer
fastq_quality_trimmer	fastx_reverse_complement
fastq_to_fasta	fastx_trimmer
fastx_artifacts_filter	fastx_uncollapse

FIGURE 1.28 FASTX-toolkit programs.

TABLE 1.4 FASTX-Toolkit Programs and Descriptions

Command Name	Description
fastq_to_fasta	converts FASTQ files to FASTA files
fastx_quality_stats	charts Quality Statistics and Nucleotide Distribution
fastx_collapse	collapses identical sequences into a single sequence
fastx_uncollapse	expands collapsed identical sequences
fastx_trimmer	trims reads in a FASTQ files (removing barcodes or noise)
fastx_renamer	renames the sequence identifiers in FASTQ/A file
fastx_clipper	removes sequencing adapters/linkers
fastx_clipping_histogram.pl	creates a Linker Clipping Information Histogram
fastq_quality_boxplot_graph.sh	creates quality boxplot
fastx_nucleotide_distribution_graph.sh	creates nucleotide distribution graph
fastx_nucleotide_distribution_line_graph.sh	creates nucleotide distribution line graph
fastx_reverse_complement	produces the Reverse-complement of each sequence
fastx_barcode_splitter.pl	splits a FASTQ/FASTA files containing multiple samples
fasta_formatter	changes the width of sequences line in a FASTA file
fastx_nucleotide_changer	converts FASTA sequences from/to RNA/DNA
fastq_quality_filter	filters sequences based on quality
fastq_quality_trimmer	trims (cuts) sequences based on quality
fastx_artifacts_filter	FASTQ/A Artifacts Filter
fastq_masker	masks nucleotides with “N” based on quality

Figure 1.28 shows the FASTX-toolkit tools in the “bin” directory after downloading and extracting the compressed archive file.

FASTX-toolkit includes several tools for the processing of FASTQ files as described in Table 1.4. You can display the usage and options of any of the executable programs by entering the program name with “-h” option on the command-line prompt. For instance, to display the help for “fastq_quality_filter”, simply enter the following on the command line:

```
fastq_quality_filter -h
```

To show how FASTQ files are processed, we will download a raw FASTQ file from the NCBI SRA database and modify its name for the practice. The following commands create the

directory “preprocessing”, then download the FASTQ file from the NCBI SRA database, and finally rename it to “bad.fastq” file for the practice purpose. The script then generates the QC FastQC report and displays the report on the Firefox browser.

```
mkdir preprocessing
cd preprocessing
fasterq-dump --verbose SRR957824
rm SRR957824_1.fastq
mv SRR957824_2.fastq bad.fastq
fqfile=$(ls *.fastq)
fastqc $fqfile
htmlfile=$(ls *.html)
firefox $htmlfile
```

When all the commands have been executed sequentially without an error, the QC report will be displayed on the Firefox Internet browser. Study the reports carefully and identify any potential problems on the quality metrics that we have discussed in the previous section. Figure 1.29 shows that the reads in the file have three failures and a single warning. Next, we will try to fix these problems as possible.

Using such FASTQ file in the downstream analysis without fixing some of the quality problems will definitely impact the results negatively and may lead to misleading results. The good strategy whenever there are warnings or failures is to try the available ways to fix the problems as possible, and if there is any unfixable problem, you may need to be aware of it and to know how it may affect the results.

Summary

- ✓ Basic Statistics
- ✗ Per base sequence quality
- ✓ Per sequence quality scores
- ✗ Per base sequence content
- ✓ Per sequence GC content
- ✓ Per base N content
- ✓ Sequence Length Distribution
- ✓ Sequence Duplication Levels
- ! Overrepresented sequences
- ✓ Adapter Content
- ✗ Kmer Content

✗ Per base sequence quality

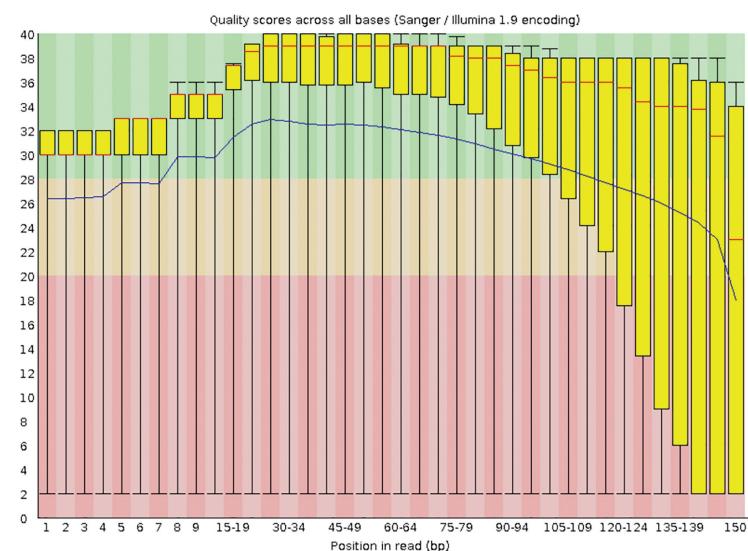


FIGURE 1.29 The QC report summary and per base sequence quality for “bad.fastq” file.

Figure 1.29 shows a common deterioration of the quality of bases toward the end of the reads produced by short-read sequencing instruments. We can also notice that some quality scores in some positions are low as 2 Phred (probability of error is 0.6).

The report shows three failures and a single warning: failed per base sequence quality (Figure 1.29), failed per base sequence content and failed k-mer content (Figure 1.30), and overrepresented sequences warning (Figure 1.31).

The QC processing strategies are different from a FASTQ file to another depending on the failed metrics. Understanding the problem always gives a good idea about which kinds of QC processing to perform. In our example file, we will begin by filtering the low-quality reads and clipping the overrepresented sequences and then we will run FastQC again to see how the quality is improved.

First, we will try to fix the per base quality score of the reads in the FASTQ file by using “fastq_quality_filter” to keep the reads that have 80% of the bases which have quality scores equal or greater than 28. The following script performs filtering (the output file is “filtered.fastq”), runs FastQC to generate the new QC report, and then runs Firefox to display the QC report on the Internet browser:

```
fastq_quality_filter \
-i bad.fastq \
-q 28 \
```

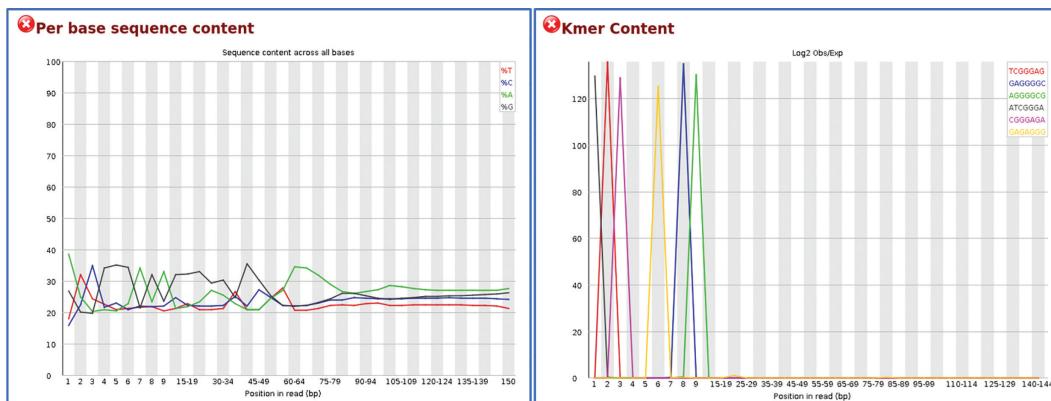


FIGURE 1.30 Failed per base sequence content and k-mer content.

Overrepresented sequences			
Sequence	Count	Percentage	Possible Source
ATCGGGAGAGGGCGGGGAGGGGAAGAGGGAGAATTGGGGGGGGCGG	2521	0.1406545093411667	No Hit
ATCGGGAGAGGGCGGGGGGGGAAGAGGGAGAATTGGGGGGGGCGG	1899	0.10595117542200538	No Hit

FIGURE 1.31 Overrepresented sequences that raised a warning.

```

-p 80 \
-o filtered.fastq \
-Q33
fastqc filtered.fastq
firefox filtered_fastqc.html

```

In the above script, “-i” option specifies the input FASTQ file, “-q” specifies the minimum Phred quality threshold, “-p” specifies the percentage of bases of the reads that have at least the specified threshold quality, “-o” specifies a name of the output FASTQ file where the filtered reads are stored, and “-Q33” is to tell the program that the FASTQ quality encoding is Phred+33 (the default is “-Q64”; therefore, we must use “-Q33” for FASTQ files with Illumina 1.9 encoding or later).

Figure 1.32 shows the per base sequence quality graph of the filtered FASTQ file. The filtering process removed 499,970 reads, which did not meet the criteria. The per base sequence quality, which is the most important metric, has been improved and per base sequence content has been also improved. However, some positions at the ends of the reads have still low Phred quality scores. We can trim the low-quality bases from the ends of the reads by using the “fastq_quality_trimmer” program. Instead of removing the reads that

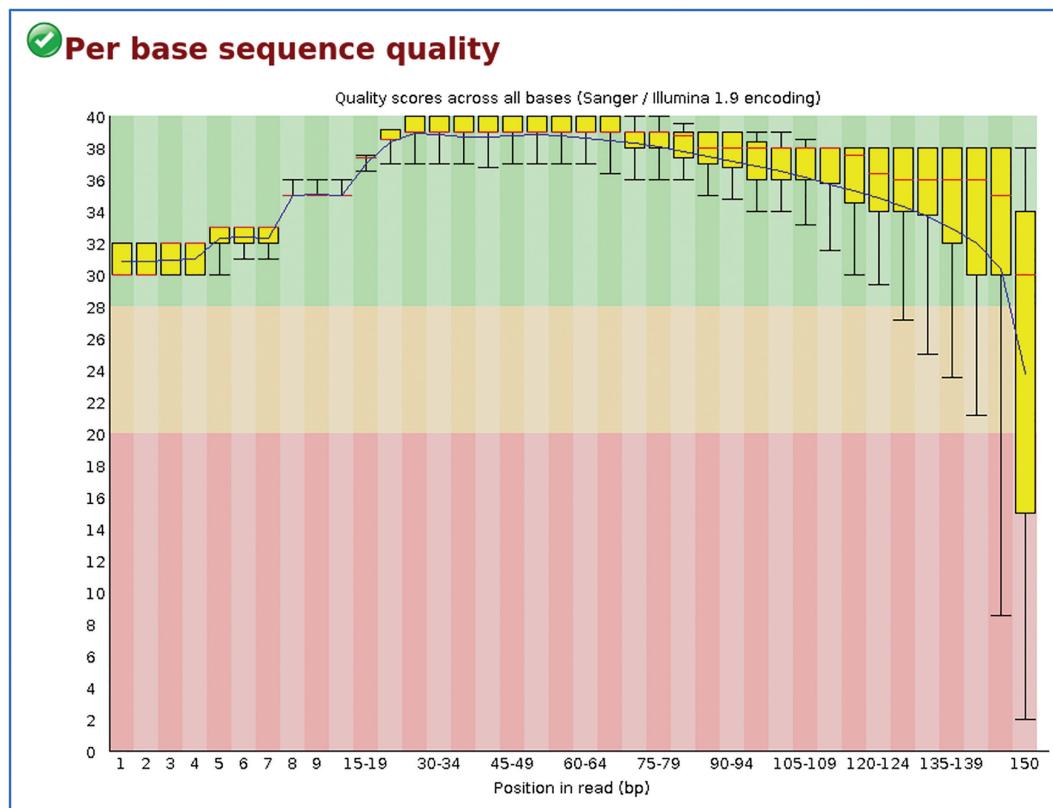


FIGURE 1.32 A graph of the filtered “bad.fastq” file with low-quality bases at the read ends.

do not meet the criteria entirely, this program cuts only the bases, whose quality scores are less than the specified threshold, from the ends of the reads.

```
fastq_quality_trimmer \
-i bad_filt.fastq \
-t 28 \
-o bad_filt_trim.fastq \
-Q33
fastqc bad_filt_trim.fastq
htmlfiles=$(ls *.html)
firefox $htmlfiles
```

The “-t” option specifies the quality threshold, which is the minimum quality score below which the bases will be trimmed from the ends of the reads. When trimming is performed, the resulted reads may be of unequal lengths, which may not be accepted by some programs used in following steps of analysis. As shown in Figure 1.33, although the per base sequence quality has been improved by trimming, it also raised a sequence length distribution warning since trimming resulted in reads with unequal lengths. We may need to filter reads by length.

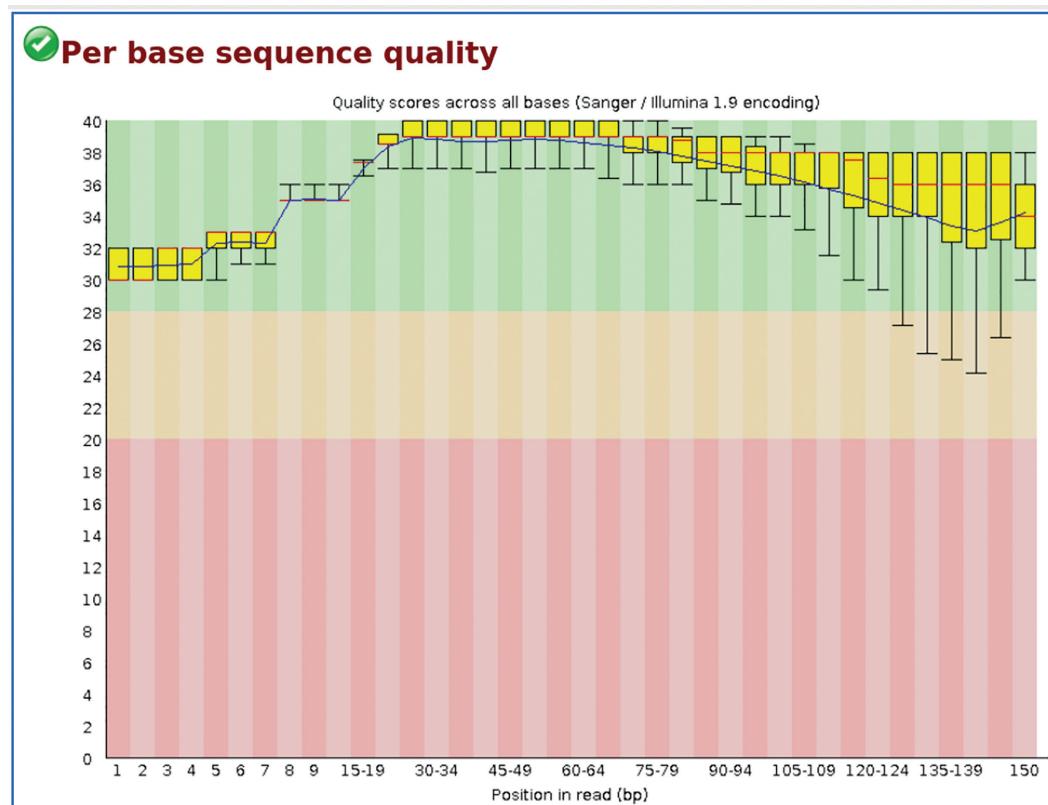


FIGURE 1.33 The QC report of the filtered and trimmed “bad.fastq” file.

The sequence length distribution warning also can be raised by clipping adaptors or overrepresented sequences. Thus, we can use “fastx_clipper” first to remove the overrepresented sequences (see Figure 1.31). The following script removes a contaminating overrepresented sequence:

```
fastx_clipper \
-a ATCGGGAGAGGGGCGGGGAGGGGAAGAGGGGAGAATTGGGGGGGGCCGG \
-i bad_filt_trim.fastq \
-o bad_filt_trim_clip.fastq \
-v \
-Q33
fastqc bad_filt_trim_clip.fastq
htmlfiles=$(ls *.html)
firefox $htmlfiles
```

Since some aligners in the next step of analysis may not accept sequences with unequal lengths, we can use a bash script to filter out the short reads. Figure 1.34 shows sequence length distribution. If the aligner that we intend to use does not accept unequal read lengths, then we can filter out all reads whose length is less than 150 bases using the following script:

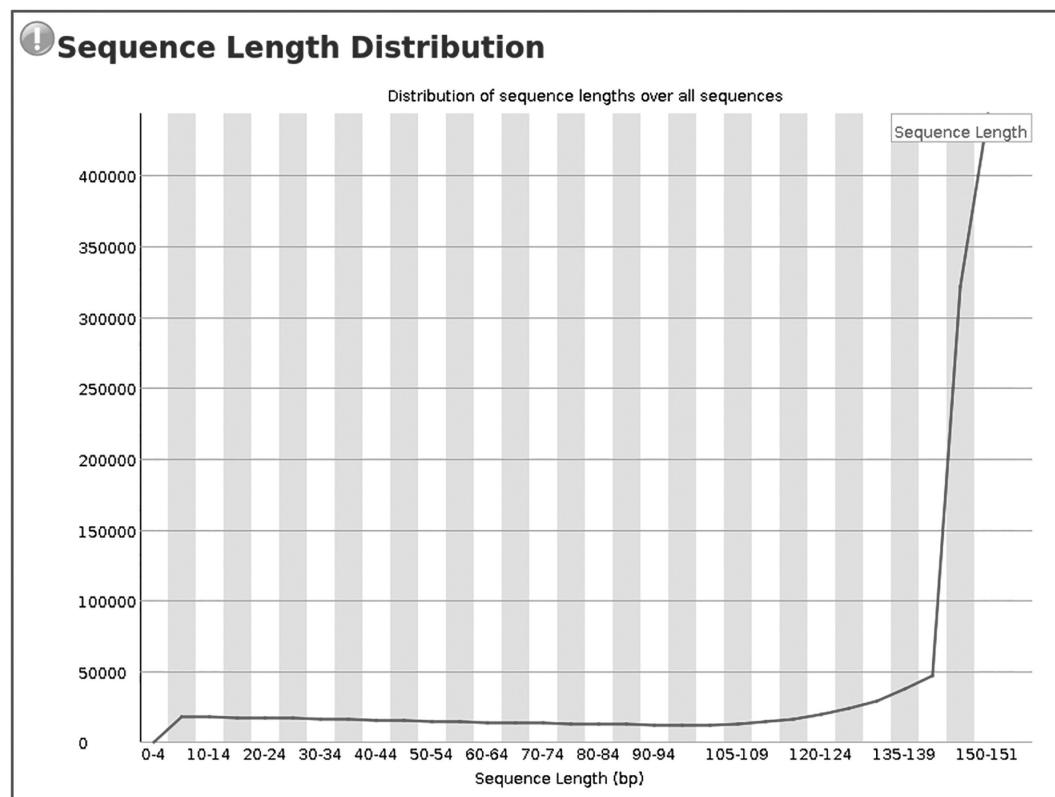


FIGURE 1.34 Sequence length distribution (different lengths).

```

paste <(cat bad_filt_trim_clip.fastq | paste - - - -) \
| awk -v FS='\t' 'length($2) >= 150 && length($4) >= 150' \
| tee >(cut -f 1-4 | tr '\t' '\n'>bad_filt_trim_clip_eq.fastq)
fastqc bad_filt_trim_clip_eq.fastq
htmlfiles=$(ls *.html)
firefox $htmlfiles

```

If you use the above script for other FASTQ files, you may need to change the read length and the numbers of the columns. In our example FASTQ file, “\$2” is for the sequence column and “\$4” is for the quality column. The numbers of these columns may vary depending on the content of the FASTQ definition line.

Figure 1.35 shows the per base sequence quality of the final FASTQ file. You should remember that you may not be able to fix all quality problems, and that filtering and clipping may compromise the sequencing depth. Fortunately, most of the problems other than base quality errors can be tolerated by the majority of the aligning programs. However, we should try to fix the failed metrics as possible before continuing to the subsequent step of the analysis.

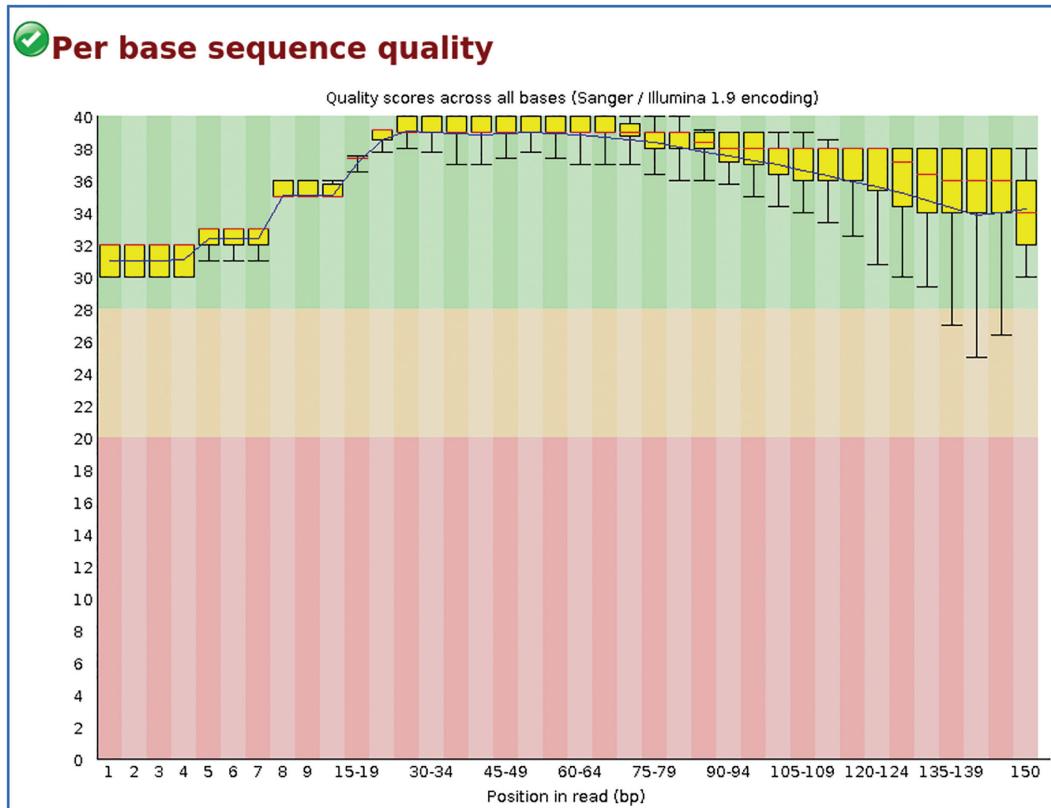


FIGURE 1.35 Cleaned FASTQ file.

The FASTX-toolkit tools listed in Table 1.4 are used for quality assessment and quality adjustment. The major limitation is that “fastq_quality_filter” of FASTX-toolkit does not process the paired-end FASTQ files together and that usually results in singletons or reads without pairs in any of the two paired-end FASTQ files. Most aligners do not accept to process paired-end FASTQ files with singletons. The FASTX-toolkit solution to the singleton problem is to mask the low-quality bases instead of removing the reads with low-quality bases. Thus, “fastq_masker” program is used instead of “fastq_quality_filter” to mask the bases of Phred quality score less than a user-defined threshold “-q”.

```
fastq_masker \
-q 20 \
-i bad.fastq \
-o bad_masked.fastq \
-Q33
fastqc bad_masked.fastq
firefox bad_masked_fastqc.html
```

The above “fastq_masker” command masks the bases with quality lower than 20 Phred quality score “-q 20”; therefore, they will be ignored by aligners and assemblers.

For paired-end FASTQ files produced by an Illumina instrument, there is another FASTQ processing program, developed by Illumina for paired-end FASTQ files, called Trimmomatic [15]. It is a multithreaded command-line Java-based program and is more modern than FASTX-toolkit. It was developed by Illumina to perform several operations, including detection and removing the known adaptor fragments (adapter.clip), trimming low-quality regions from the beginning of the reads (trim.leading), trimming low-quality regions from the end of the reads (trim.trailing), filtering out short reads (min.read.length), in addition to other operations with different quality-filtering strategies for dropping low-quality bases in the reads (max.info and sliding.window). Trimmomatic can be used in two modes: simple and palindrome modes. In the simple mode, for removing adaptor sequences, the pairwise local alignment between adaptor sequence and reads is used to scan reads from 5' ends to 3' ends using *seed and extend* approach. If a score of a match exceeds a user-defined threshold, both the matched region and the region after alignment will be removed. The entire read is removed if an alignment covers all the read. The simple Trimmomatic approach may not be able to detect the short adaptor sequence. Therefore, the palindrome model is used because it is able to detect and remove short fragment sequences of adaptors. Palindrome is used only for the paired-end data. Both forward and reverse reads will have equal number of valid bases and each read complements another. The valid reads are followed by the contaminating bases from the adaptors. The tool uses the two complementary reads to identify the adaptor fragment or any other contaminating technical sequence by globally aligning the forward and reverse reads. An alignment score that is greater than a user-defined threshold indicates that the first parts of each read reversely complement one another and the remaining read fragments which match the adaptor sequence will be removed.

Trimmomatic is available at “<http://www.usadellab.org/cms/index.php?page=trimmomatic>”. You can download it from the website and unzip it using the following script:

```
$ wget http://www.usadellab.org/cms/uploads/supplementary/
Trimmomatic/Trimmomatic-0.39.zip
$ unzip Trimmomatic-0.39.zip
```

Notice that the version may change in the future. The unzipped directory is “Trimmomatic-0.39”, where there will be two files (“LICENSE” and “trimmomatic-0.39.jar”) and a directory (“adapters”). The file “trimmomatic-0.39.jar” is the Java executable program that performs the preprocessing tasks and the directory “adapters” contains the known adaptor sequences in FASTA files. The following script uses Trimmomatic to reprocess the paired-end FASTQ files, then runs FastQC to generate QC reports, and finally displays the reports on the Firefox browser:

```
java -jar ../Trimmomatic-0.39/trimmomatic-0.39.jar \
PE SRR957824_1.fastq SRR957824_2.fastq \
out_PE_SRR957824_1.fastq out_UPE_SRR957824_1.fastq \
out_PE_SRR957824_2.fastq out_UPE_SRR957824_2.fastq \
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10:2:True \
LEADING:3 \
TRAILING:3 \
ILLUMINACLIP:TruSeq2-PE.fa:2:30:10 \
SLIDINGWINDOW:5:30 \
MINLEN:35
fastqc out_PE_SRR957824_1.fastq out_PE_SRR957824_2.fastq
firefox out_PE_SRR957824_1_fastqc.html out_PE_SRR957824_2_fastqc.
html
```

The option “PE” is used for paired end, and then the two paired-end FASTQ files “SRR957824_1.fastq” and “SRR957824_2.fastq” were provided as inputs. The adaptors that were detected and removed from the reads are stored in the “TruSeq3-PE.fa” file in the “adapters” directory. Hence, “ILLUMINACLIP:TruSeq2-PE.fa” is used to specify the file in which the adaptor sequences are stored. The program removed the leading and trailing edges of reads with low quality that is below 3 Phred quality score. The “SLIDING-WINDOW:5:35” is used so that the program can scan the read with a 5-base wide sliding window and remove a read when the window per base average quality score declines to below 30. Finally, the program removes the reads that are shorter than 35 bases.

In Figures 1.36 and 1.37, notice how the quality of the two files have been improved and also notice that the total sequence is equal in both files. However, the read lengths vary.

If for any reason we need reads of the same length as some aligners may require, we can set “MINLEN:” to the maximum length. Since the maximum read length is 150 bases, we can use “MINLEN:151” as follows:

```
java -jar ../Trimmomatic-0.39/trimmomatic-0.39.jar \
PE SRR957824_1.fastq SRR957824_2.fastq \
out_PE_SRR957824_1.fastq out_UPE_SRR957824_1.fastq \
out_PE_SRR957824_2.fastq out_UPE_SRR957824_2.fastq \
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10:2:True \
LEADING:3 \
TRAILING:3 \
ILLUMINACLIP:TruSeq2-PE.fa:2:30:10 \
SLIDINGWINDOW:5:30 \
MINLEN:150
fastqc out_PE_SRR957824_1.fastq out_PE_SRR957824_2.fastq
firefox out_PE_SRR957824_1_fastqc.html out_PE_SRR957824_2_fastqc.html
```

Check out the reports of the two paired-end FASTQ files to see that only the reads with equal length are left.

There are several other programs that can be used, as well, for the quality improvement of FASTQ reads. For instance, Fastp can be used for the filtering of low-quality reads and

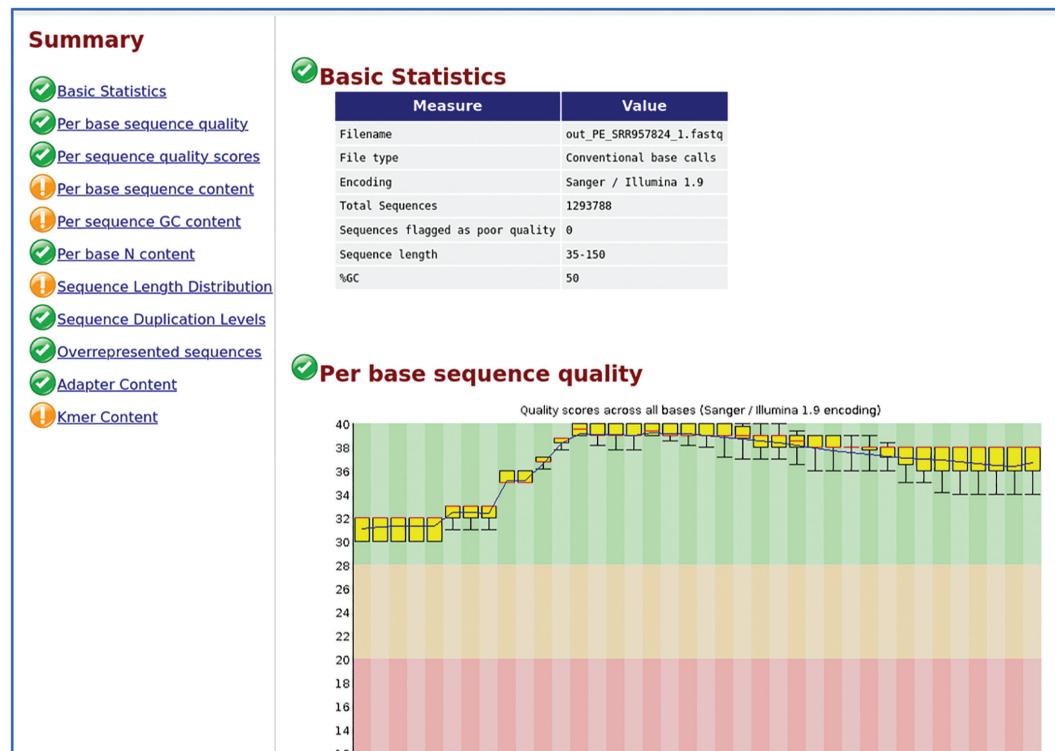


FIGURE 1.36 Trimmomatic processed forward FASTQ file.

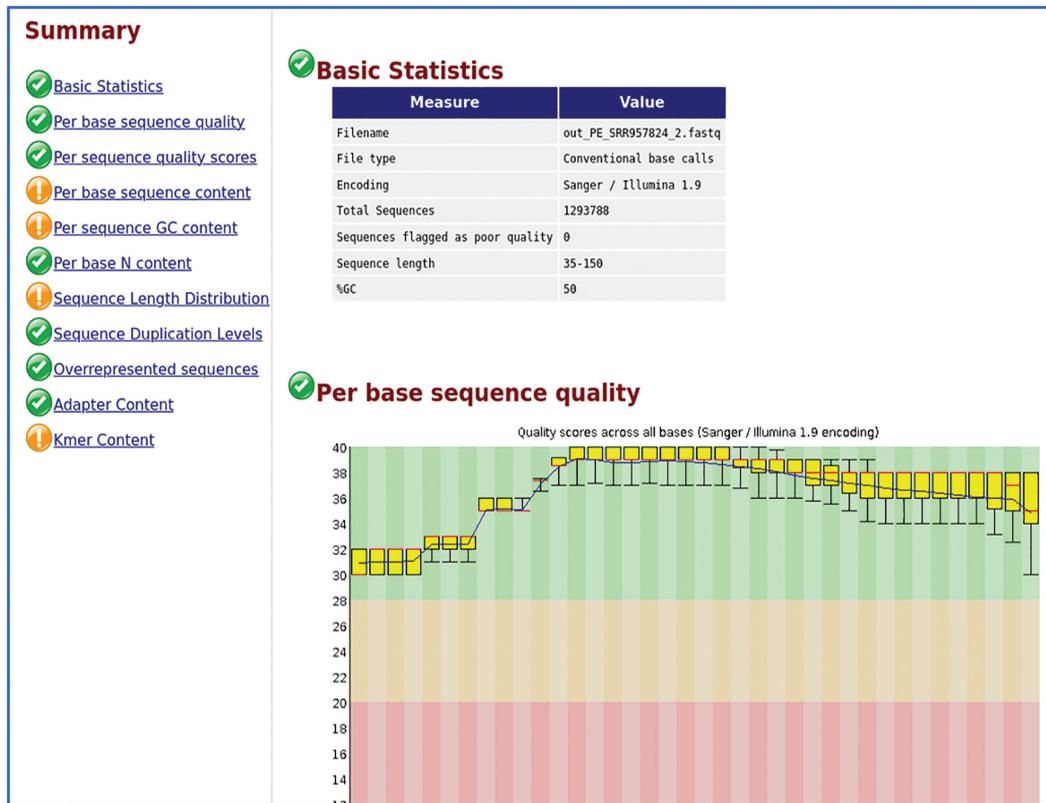


FIGURE 1.37 Trimmomatic processed reverse FASTQ file.

adaptor trimming. This program is specifically fast and easy to use as part of a pipeline. Moreover, it is able to identify adaptor sequences and trim them without the need of providing adaptor sequences [16].

1.7 SUMMARY

The NGS produces short reads that are widely used for the different sequencing applications for the high accuracy and low cost. However, the long reads produced by the TGS (Pacific Bioscience and Oxford Nanopore Technologies) have also gained some popularity in applications like *de novo* assembly, metagenomics, and epigenetics. The accuracy of the long-read technologies has been substantially improved, but the cost is still high and less affordable when they are compared to short-read technologies. The sequencing depth and base call quality are the two crucial factors for most applications, and the analysts must keep looking at them before proceeding with the analysis. Most HTS instruments perform quality control before delivering raw sequence data in FASTQ files. However, per base qualities and other quality metrics must be assessed before using raw data in any analysis.

There are several programs for quality assessment, but FastQC is the most popular one. FastQC is a user-friendly program to assess the quality of the reads generated by any of the sequencing technologies, and it produces a report that summarizes the results in graphs that are easy to interpret. The potential quality problems include low-quality bases, presence of adaptor sequences connected to the reads, presence of adaptor dimers or other technical contaminating sequences, overrepresented PCR sequences, sequence length distribution, per base sequence content, per sequence GC content, per base N content, and k-mer content. The per base sequence quality and adaptor content are the most important metrics that we should look at and take the appropriate action. The ideal sequencing data are the one without warnings or failed metrics. Therefore, we should try to fix the problems as possible. However, some problem may not be solved. If the unsolved problem does not affect the reads severely, that data still can be used in the analysis. However, we must be aware that unsolved problems may have some negative impact in the results. The read quality problems can be solved based on the failed metrics by removing low-quality reads, trimming the reads from the beginning and the end of the reads, and masking the bases with low-quality scores. There are several programs for the processing of raw sequence data. FASTX-toolkit is the most popular one for single-end FASTQ files, and Trimmomatic is more sophisticated and can be used for both single-end and paired-end raw data. Fastp filters low-quality reads and automatically recognizes and trims adaptor sequences. It is important to process the paired-end FASTQ files (forward and reverse) together to avoid leaving out singlettons, which may not be accepted by almost all aligners. In this chapter, we discussed the command-line programs for quality controls. However, those programs or similar ones are implemented in Python, R, and other programming languages, but understanding the general principle for checking the raw data quality and solving potential quality problems are the same. Most sequencing applications use these kinds of QC processing, but when we cover the metagenomic data analysis, you will learn how to preprocess microbial raw data using different programs. Once the raw sequencing data are cleaned, then we can move safely to the next step of sequence data analysis depending on the application workflow that we are adopting.

REFERENCES

1. Holley RW, Everett GA, Madison JT, Zamir A: Nucleotide sequences in the yeast alanine transfer ribonucleic acid. *J Biol Chem* 1965, 240: 2122–2128.
2. Jou WM, Haegeman G, Ysebaert M, Fiers W: Nucleotide sequence of the gene coding for the bacteriophage MS2 coat protein. *Nature* 1972, 237(5350):82–88.
3. Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen Y-J, Chen Z et al: Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 2005, 437(7057):376–380.
4. Braslavsky I, Hebert B, Kartalov E, Quake SR: Sequence information can be obtained from single DNA molecules. *Proceedings of the National Academy of Sciences* 2003, 100(7):3960–3964.
5. Rhoads A, Au KF: PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics* 2015, 13(5):278–289.
6. Levene MJ, Korlach J, Turner SW, Foquet M, Craighead HG, Webb WW: Zero-mode waveguides for single-molecule analysis at high concentrations. *Science* 2003, 299(5607):682–686.

7. Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM: The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res* 2010, 38(6):1767–1771.
8. FASTQ Files [https://support.illumina.com/help/BaseSpace_O LH_009008/Content/Source/Informatics/BS/FASTQFiles_Intro_swBS.htm]
9. Leinonen R, Sugawara H, Shumway M: The sequence read archive. *Nucleic Acids Res* 2011, 39(Database issue):D19–21.
10. Andrews S: *FastQC: A Quality Control Tool for High Throughput Sequence Data*. Babraham Bioinformatics, Babraham Institute, Cambridge, United Kingdom; 2010.
11. Chen Y-C, Liu T, Yu C-H, Chiang T-Y, Hwang C-C: Effects of GC bias in next-generation-sequencing data on de novo genome assembly. *PLOS One* 2013, 8(4):e62856.
12. Lightfield J, Fram NR, Ely B: Across bacterial phyla, distantly-related genomes with similar genomic GC content have similar patterns of amino acid usage. *PLoS One* 2011, 6(3):e17677.
13. Romiguier J, Ranwez V, Douzery EJ, Galtier N: Contrasting GC-content dynamics across 33 mammalian genomes: relationship with life-history traits and chromosome sizes. *Genome Res* 2010, 20(8):1001–1009.
14. FASTX-toolkit [http://hannonlab.cshl.edu/fastx_toolkit/]
15. Bolger AM, Lohse M, Usadel B: Trimmomatic: A flexible trimmer for Illumina sequence data. *Bioinformatics* 2014, 30(15):2114–2120.
16. Chen S, Zhou Y, Chen Y, Gu J: fastp: An ultra-fast all-in-one FASTQ preprocessor. *Bioinformatics* 2018, 34(17):i884–i890.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Mapping of Sequence Reads to the Reference Genomes

2.1 INTRODUCTION TO SEQUENCE MAPPING

So far, we have already gone through the first two steps of the NGS/HTP data analysis, namely acquiring the raw data in FASTQ file format and read quality control. Up to this point, you know that the sequencing raw data must be cleaned from errors and artifacts, as much as possible, before moving on to the next step of the data analysis. This chapter discusses the alignment of reads (short or long) to a reference genome of an organism. This step is crucial for most of the sequencing applications including reference-guided genome assembly, variant discovery, gene expression (RNA-Seq), epigenetics (ChIP-Seq, Methyl-Seq), and metagenomics (targeted and shotgun). The reference genome sequence of an organism is a key element of read alignment or mapping. Scientists have devoted enormous amount of time and efforts to determine the sequences of many organisms. Complete genomes of hundreds of organisms have already been sequenced and the list continues to grow. The sequencing of human genome was completed in 2003 by the National Human Genome Research Institute (NHGRI), followed by sequencing the genomes of a variety of model organisms that are used as surrogates in studying the human biology, then genomes of numerous of organisms, including some extinct organisms like Neanderthals, were sequenced. The first sequenced genomes of model organisms include the rat, puffer fish, fruit fly, sea squirt, roundworm, and the bacterium *Escherichia coli*. The NHGRI has sequenced numerous species with the aim to provide data for understanding genetic variations among organisms. Genome sequences are available in sequence databases funded by governments and supported by institutions. A reference genome sequence of an organism is a curated sequence that represents the genome of the individuals of that organism. However, the sequences of the individuals are varied and the reference sequence is only a sequence that we compare other sequences to. These days, there are reference genomes for thousands of organisms, including animal, plants, fungi, bacteria, archaea, and viruses,

together with their gene annotations, which can be used in the process of read alignment/mapping to act as guides on which new genomes are assembled fast. A reference genome of an organism is a curated sequence that is built up using the DNA information of several normal individuals of that organism. The reference genome curation was pioneered by the Genome Reference Consortium (GRC), which is founded in 2008 as a collaboration of the National Center for Biotechnology Information (NCBI), the European Bioinformatics Institute (EBI), the McDonnell Genome Institute (MGI), and the Wellcome Sanger Institute to maintain and update the human and mouse genome reference assemblies. Now, GRC maintains the human, mouse, zebrafish, rat, and chicken reference genomes. Reference genomes of other organisms are curated by specialized institutions including NCBI and many others, which manually select genome assemblies that are identified as standard or representative sequences (RefSeq) against which data of the individuals from those organisms can be compared. All eukaryotes have a single reference genome per species, but prokaryotes may have multiple reference genome sequences for a species. The NCBI curates reference genomes from the assemblies categorized as RefSeq on the GenBank database. If a eukaryotic species has no assemblies in the RefSeq, then the best GenBank assembly for that species is selected as a representative genome. Viruses as well may have more than one reference genome per species. Generally, the update of a reference genome of any species is a continuous process and a new version, usually called “Build”, may be released whenever new information emerges. A release of a reference genome may be accompanied by gene annotations. A well-curated reference genome, like human and other model organisms’ reference genomes, is usually released with annotation information such as gene annotation and variant annotation. Reference genomes are made available at the NCBI website in both FASTA file format and GenBank file format. Several annotation files may be

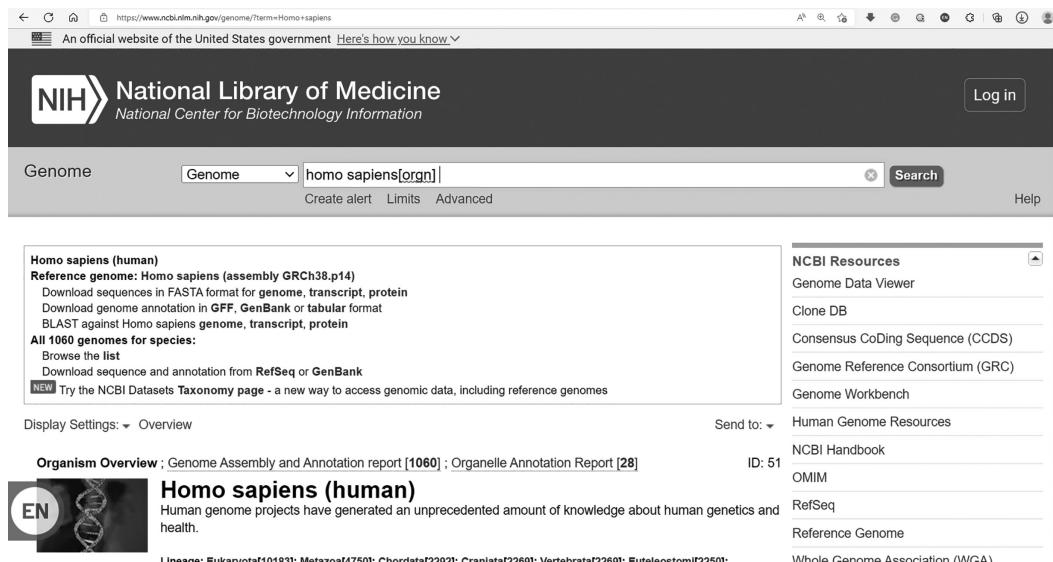


FIGURE 2.1 Human reference genome on the NCBI Genome page.

found including gene annotation in GFF/GTF file format, GenBank format, and tabular format. The reference transcriptome (whole mRNA of an organism) and proteins may also be available as shown in Figure 2.1.

For the alignment/mapping of reads produced by sequencing instruments, we may need to download a reference genome of the species from which the sequencing raw data are taken. The sequence of the reference genome must be in the FASTA file format. For example, to download the FASTA file of the human genome, you can copy the link from “genome” hyperlink on the Genome database web page and on Linux terminal use “wget” to download the file to the directory of your choice “e.g. refgenome”:

```
mkdir refgenome
wget \
-O "refgenome/GRCh38.p13_ref.fna.gz" \
https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/405/
GCF_000001405.39_GRCh38.p13/GCF_000001405.39_GRCh38.p13_genomic.
fna.gz
```

This script will create the “refgenome” directory, where it will download the compressed FASTA sequence of the human reference genome “GRCh38.p13_ref.fna.gz”. The size of the compressed current FASTA sequence file of the human genome (GRCh38.p13) is only 921M. We can decompress it using the “gunzip” command.

```
gunzip -d GRCh38.p13_ref.fna.gz
```

This command will decompress the reference genome file to “GRCh38.p13_ref.fna” and the file size now is 3.1G. A large file can be displayed using a program for displaying a large text such as “less” or “cat” Linux commands. The reference sequences are in the FASTA file format. A file contains several sequences representing the genomic units such as chromosomes. Each FASTA sequence entry consists of two parts: a definition line (defline), which is a single line that begins with “>” symbol, and a sequence, which may span several lines. Figure 2.2 shows the beginning of the human genome reference sequence. Notice that the defline includes the GenBank accession of the sequence, species scientific name, genome unit (chromosome number), and the human genome Build. A chromosome sequence may begin with multiple ambiguous bases (Ns). In Figure 2.2, we removed several lines of Ns intentionally to show the DNA nucleobases.

The following Unix/Linux commands are used with the text files as general and here we can use them with FASTA files to collect some useful information.

To display the FASTA file content page by page, you can use “less” command:

```
less GRCh38.p13_ref.fna
```

To count the number of FASTA sequences in the FASTA file, use “grep” command:

```
grep -c ">" GRCh38.p13_ref.fna
```

FIGURE 2.2 Part of the FASTA sequence of the human reference genome

To count the total number of bases in the reference file, you can combine “grep”, “wc”, and “awk” commands as follows:

```
grep -v ">" GRCh38.p13_ref.fna | wc | awk '{print $3-$1}'
```

If for any reason, you want to split the reference sequences into files, you can use the following script that creates the directory, “chromosomes”, and then it splits the main FASTA file into several FASTA files:

```
mkdir chromosomes
cd chromosomes
csplit -s -z ./GRCh38.p13_ref.fna '/>/' '{*}'
for i in xx* ; do \
n=$(sed 's/>// ; s/ .*// ; 1q' "$i") ; \
mv "$i" "$n.fa" ; \
done
```

The annotation files relevant to a reference genome may also be needed for some of the steps in the downstream analysis. You can download the annotation file as above. The annotation file is a description of where genetic element also called a feature such as genes, introns, and exons are located in the genome sequence, showing the start and end coordinates, and feature name. The annotation files are usually in GFF or GTF file format. The GFF (General feature format) is a simple tab-delimited text file for describing genomic features and mapping them to the reference sequence in the FASTA file. The GTF (Gene Transfer Format) is similar to GFF but it has additional elements. Figure 2.3 shows the first part of the human annotation file in the GFF format. The content of an annotation file including the chromosome name or chromosome GenBank accession in the first column, and features and other annotations are in the other columns.

Both the FASTA reference file and (sometimes) its annotation file are required by the alignment programs, shortly called aligners, for mapping the reads in the FASTQ files to

```

##gff-version 3
#!gff-spec-version 1.21
#!processor NCBI_annotwriter
#!genome-build GRCh38.p14
#!genome-build-accession NCBI_Assembly:GCF_000001405.40
#!annotation-source NCBI Homo sapiens Annotation Release 110
##sequence-region NC_000001.11 1 248956422
##species https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=9606
NC_000001.11 RefSeq region 1 248956422 + .
NC_000001.11 BestRefSeq pseudogene 11874 14409 . .
NC_000001.11 BestRefSeq transcript 11874 14409 . +
NC_000001.11 BestRefSeq exon 11874 12227 . + .
NC_000001.11 BestRefSeq exon 12613 12721 . + .
NC_000001.11 BestRefSeq exon 13221 14409 . + .
NC_000001.11 BestRefSeq pseudogene 14362 29370 . - .
NC_000001.11 BestRefSeq transcript 14362 29370 . - .
NC_000001.11 BestRefSeq exon 29321 29370 . - .
NC_000001.11 BestRefSeq exon 24738 24891 . - .
NC_000001.11 BestRefSeq exon 18268 18366 . - .
NC_000001.11 BestRefSeq exon 17915 18061 . - .
NC_000001.11 BestRefSeq exon 17606 17742 . - .
NC_000001.11 BestRefSeq exon 17233 17368 . - .
NC_000001.11 BestRefSeq exon 16858 17055 . - .
NC_000001.11 BestRefSeq exon 16607 16765 . - .
NC_000001.11 BestRefSeq exon 15796 15947 . - .
ID=NC_000001.11:1..248956422
ID=gene-DDX11L1;Dbx>
ID=rna-NR_046018.2-1;Paren>
ID=exon-NR_046018.2-2;Paren>
ID=exon-NR_046018.2-3;Paren>
ID=gene-WASH7P;Dbxr>
ID=rna-NR_024540.1-1;Paren>
ID=exon-NR_024540.1-2;Paren>
ID=exon-NR_024540.1-3;Paren>
ID=exon-NR_024540.1-4;Paren>
ID=exon-NR_024540.1-5;Paren>
ID=exon-NR_024540.1-6;Paren>
ID=exon-NR_024540.1-7;Paren>
ID=exon-NR_024540.1-8;Paren>
ID=exon-NR_024540.1-9;Paren>

```

FIGURE 2.3 Part of the human annotation file in GTF file format.

the reference genome in a process known as read sequence mapping or alignment. In the read mapping process, the FASTA files may contain millions of read sequences that we wish to align to a sequence of a reference genome to produce aligned reads in a file format called SAM, which stands for Sequence Alignment Map format. The aligned reads can also be stored in the SAM binary form called BAM (Binary Alignment Map format). We will discuss this file format later in some detail.

In general, sequence mapping or alignment requires three elements: A reference file in the FASTA format, short-sequence reads in FASTQ files, and an aligner, which is a program that uses an algorithm to align reads to a reference genome sequence. We have already discussed how to download the sequence of a reference genome of an organism from the NCBI Genome database. However, before using a reference genome with any aligner, it may require indexing with the “samtools faidx” command. You can download and install Samtools by following the instructions available at “<http://www.htslib.org/download/>”. On Ubuntu, you can install it using the following command:

```
sudo apt-get install samtools
```

Once you have installed Samtools successfully, you can use that tool to index the reference genome and other tasks that you will learn later.

You have already downloaded the human reference genome above. If you didn’t do that, you can download and decompress it using the following commands:

```

mkdir refgenome
wget \
-O "refgenome/GRCh38.p13_ref.fna.gz" \
https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/405/GCF\_000001405.39\_GRCh38.p13/GCF\_000001405.39\_GRCh38.p13\_genomic.fna.gz
cd refgenome
gunzip -d GRCh38.p13_ref.fna.gz

```

Notice that the name of the reference genome or its URL may change in the future. The above commands will create the directory “refgenome” where it downloads the human reference genome and decompresses it. Once the reference genome has been downloaded, you can use “samtools faidx” to index it as follows:

```
samtools faidx GRCh38.p13_ref.fna
```

To read more about this command, run “samtools faidx -h”.

Indeed, for the reference sequence to be indexed by “samtools faidx” command, it must be in FASTA format and well-formatted, which means that the FASTA sequences contained in the file must have a unique name or ID in the FASTA defline and the sequence lines of each sequence must be of the same length. Indexing a reference genome with Samtools enables efficient access to arbitrary regions within the FASTA file of the reference sequence.

The above “samtools faidx” command creates an index file “GRCh38.p13_ref.fna.fai” for the reference genome with the same name as that of the reference genome but with “.fai” appended to the file name. For the FASTA file, an fai index file is a text file consisting of lines, each with five TAB-delimited columns, including NAME (name of this reference sequence), LENGTH (length of sequence), OFFSET (sequence’s first base in bytes), LINEBASES (the number of bases on each line), and LINEWIDTH (the number of bytes in each line) as shown in Figure 2.4.

Remember that before you use a reference genome with any aligner, you must index it with “samtools faidx” as above, and the FASTA file and the index file must be in the same directory. In some reference genome sequence, the sequence names are labeled by chromosomes (e.g., chr1) instead of accession numbers.

In the following, we will discuss the commonly used algorithms for read alignments and the popular aligners.

NT_187396.1	2274	3133677187	80	81
NT_187397.1	2165	3133679590	80	81
NT_187398.1	1942	3133681883	80	81
NT_187399.1	1472	3133683950	80	81
NT_187400.1	21476	3133685541	80	81
NT_187401.1	4416	3133707386	80	81
NT_187402.1	1201	3133711958	80	81
NT_187403.1	1444	3133713275	80	81
NT_187404.1	2276	3133714838	80	81
NT_187405.1	998	3133717243	80	81
NT_187406.1	12399	3133718354	80	81
NT_187407.1	37690	3133731008	80	81
NT_187408.1	1179	3133769270	80	81

FIGURE 2.4 Part of the fai index file of the human reference genome.

2.2 READ MAPPING

Sequence alignment is one of the most important tasks in bioinformatics. It has been used for decades before the recent revolution in genomics which erupted after the discovery of the high-throughput sequencing technologies. It has been used in numerous bioinformatics applications. The basic alignment algorithms are the global sequence alignment using Needleman-Wunsch [1, 2] algorithm and the local sequence alignment using Smith-Waterman (SW) [3] algorithm. Both these two legacy algorithms use *dynamic programming*, which was invented by the mathematician Richard Bellman in early 1950s. The dynamic programming in sequence alignment, which has nothing to do with computer programming, is an algorithm that attempts to find the optimal alignment of two sequences by dividing the whole alignment into smaller sub-alignments. The *global sequence alignment* is to align two sequences from end to end, and therefore, gaps can be introduced to stretch the sequences to be equal in length. It is obvious that this type of alignment does not work for aligning the massive number of short sequences to a genome sequence that may span millions of bases. The second type is the *local sequence alignment*, which aligns two sequences only to the regions of similarity. It is quite clear that this kind of sequence alignment will work for aligning reads to a reference genome. The local sequence alignment in its crude form is exhaustive in a way that if you have a single sequence and you want to align it to multiple sequences, the algorithm will exhaust all possible alignments and then report the results. In most cases, this approach is not practical for aligning millions of reads and it is computationally expensive with that massive number of genomics sequences. The Basic Local Alignment Search Tool or BLAST [4] is heuristic version of the SW local sequence alignment algorithm, which is more time-efficient than the exhaustive approach because it searches only for the most significant patterns in the sequences and performs sequence alignment accordingly. BLAST is fast but it may skip some alignments because of its heuristic nature. Now after all, we can ask the question: Is BLAST suitable for aligning millions of short-read sequences to a reference genome that may span millions of bases? To provide a practical answer to this question, assume that BLAST could align a single read to the reference genome in a second; if we have 20 million reads, BLAST will align them to the reference genome in 20 million seconds (231.5 days), if we ignore the possibility of the power or Internet outage or any other natural disaster. In conclusion, BLAST is not efficient for read sequence alignment. Considering the large size of the reference genome sequences and the large number of reads, one of the biggest challenges to perform searching and locating the mapped region with high accuracy is *indexing*. Efficient indexing algorithms are needed to organize the sequence of the reference genome and the short reads in memory efficiently and to facilitate fast searching for patterns. Computer scientists play a key role in developing data structures that allow computer programs to store and process data effectively with less computational costs. Efficient data structures will help in both indexing the sequence of a reference genome and fast searching of a read in an indexed sequence with efficient memory usage. Several data structures for indexing were proposed and implemented on software packages for read sequence alignment. The most commonly used data structures for indexing include *suffix tree*, *suffix array*, *Burrows-Wheeler*

transform (BWT), and *Full-text Minute-space* (FM-index). Aligners implement a variety of searching algorithms to determine where short reads originated in the indexed reference genome sequence. Aligning of a read to a genomic location depends on the sequence similarity. However, a good aligner should expect mismatches due to the sequencing errors and genetic variation between the reference genome and a sequenced individual. In the following, we will discuss these commonly used data structures for indexing and popular searching methods.

2.2.1 Trie

A trie or a prefix tree is a data structure that is used for fast retrieval on large datasets such as looking up sequencing reads. The name was derived by E. Fredkin in 1960 from the phrase “Information Retrieval” by Fredkin (1960) [5]; however, the idea was first described by the Norwegian mathematician Axel Thue in 1912. A trie is an ordered tree that can be used to represent a set of strings (reads) over a finite alphabet set, which is A, C, G, and T for the DNA sequences. It allows reads with common prefixes to use that prefix and stores only the ends of the reads to indicate different sequences. A trie is represented by nodes and edges. The root node is empty and then each node in the trie represents a single character. The maximum number of possible children of a node is four in case of DNA. The children of a node are ordered alphabetically. The leaf nodes are the nodes that represent the last characters of reads. A read is represented by the path from the root node to a leaf node.

The trie data structure, as it is, is not suitable for indexing a genome sequence since it stores a set of string. However, the generalized idea of the trie is used in the suffix tree which is widely used to index a reference genome sequence.

2.2.2 Suffix Tree

The suffix tree, as a generalization of the trie data structure, is basically used for pattern matching and finding substrings in a given string of characters. It is constructed as key-value pairs (as the python dictionary) where all possible suffixes of the reference sequence are the keys and positions (indexes) in the sequence as their values. In 1995, Esko Ukkonen proposed a linear-time algorithm called Ukkonen’s algorithm [6], which constructs a suffix tree in a linear time complexity that the time taken for the indexing increases linearly with the increase in the number of nodes $O(n)$.

For the sake of simplicity, we will try to show you how to build a suffix tree for a reference sequence and how mapping of the reads is carried out. Assume that our reference sequence consists of 10 bases as “CTTGGCTGGA\$”, where the positions of the bases are 0, 1, 2, ..., 9 and \$ is an empty trailing position. The first step of constructing a suffix tree is by forming suffixes (keys) and indexes (values) pairs from the sequence. We begin from the last character in the sequence, which is the empty character “\$” in the position 10 in the sequence. Then, we form the suffix “A\$”, whose first character is in position 9 in the sequence. We continue this way until all possible suffixes are created as shown below:

```

$ 10
A $ 9
GA $ 8
GGA $ 7
TGGA $ 6
CTGGA $ 5
GCTGGA $ 4
GGCTGGA $ 3
TGGCTGGA $ 2
TTGGCTGGA $ 1
CTTGGCTGGA $ 0

```

Notice that each line includes a suffix (key) and a position (value). Then from the key-value pairs, we can construct a tree made up of nodes and edges. The positions (values) will be the nodes and suffixes (keys) will be the edges of the tree. The suffix tree is built as shown in Figure 2.5, starting from the first suffix on the top and it moves down to make branched nodes and edges to avoid repeating common characters. This way, we will construct a suffix tree with nodes and edges. An entire reference genome can be divided into suffixes and stored this way with both suffixes and indexed positions in the unbranched nodes so finding a pattern or a position of a read in the reference genome will be easy.

Once the reference sequence is indexed using the suffix tree, one of several searching algorithms can be used to find the location where a read maps. For instance, to find “TGG” in Figure 2.5, we will start searching from the root looking for “T”, and from the next node, we will look for “GG”; thus, since there are two leaf nodes with the indexes 2 and 6, that means “TGG” is aligned to the reference sequence in the positions 2 and 6 as shown by the red color (Figure 2.5).

2.2.3 Suffix Arrays

The suffix array (SA) is similar to the suffix tree for pattern matching and finding substrings (reads) and it can be constructed from the suffix tree. It is basically a sorted array

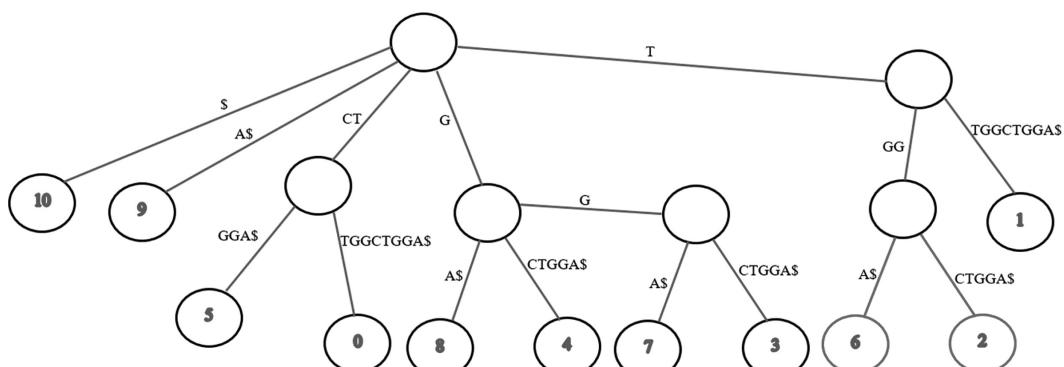


FIGURE 2.5 Nodes and edges of the suffix tree.

SA	SA
1 CTTGGCTGGA\$	11 \$
2 TTGGCTGGA\$	10 A\$
3 TGGCTGGA\$	6 CTGGA\$
4 GGCTGGA\$	1 CTTGGCTGGA\$
5 GCTGGA\$	9 GA\$
6 CTGGA\$	5 GCTGGA\$
7 TGGGA\$	8 GGA\$
8 GGA\$	4 GGCTGGA\$
9 GA\$	7 TGGGA\$
10 A\$	3 TGGCTGGA\$
11 \$	2 TTGGCTGGA\$

Sorting →

FIGURE 2.6 Suffix array.

of all suffixes of a given sequence. There is a variety of algorithms used for constructing the suffix array implemented by software packages [7]. In its simplest form, a suffix array is constructed for a sequence or a string. For the string (sequence) S with a length N and the characters (bases) indexed as 1, 2, 3, ..., N , we can construct an array for all suffixes or substrings of the sequence as $S[1\dots N]$, $S[2\dots N]$, ..., $S[N\dots N]$ and then sort the suffixes lexicographically. For instance, for the sequence $S = \text{"CTTGGCTGGA\$"}$, we can construct the arrays of suffixes and sort them as shown in Figure 2.6.

Figure 2.6 shows how the suffix array is constructed from sorted suffixes. The numbers are the positions in the sequence. The sorting of suffixes lets suffixes beginning with the same string of characters to appear one after the other and that allows a fast lookup when we try to find exact matches of a read (substring). For instance, the exact match for the reads “TGG” can be found by jumping to the sorted suffixes that begin with “TGG” and we can fast locate the positions 7 and 3 (the coordinate begins from 1 and not 0 as in suffix tree). When a reference genome is indexed using the suffix array, finding a position of a pattern or a read will have a linear time complexity. A major drawback of using suffix arrays is that they require a large memory storage depending on the size of the genome being used. STAR [8] is as an example of the aligners that use suffix arrays.

2.2.4 Burrows–Wheeler Transform

The Burrows–Wheeler Transform or shortly BWT is a data structure algorithm that transforms a string (sequence) into a compressible form that allows fast searching. The BWT is used by BWA [9], Bowtie [10], and other aligners. BWT algorithm has been used by Unix and Linux for data compression as bzip2 compression utility.

Let S be a sequence of characters of a sequence or string of a length n . For a genomic sequence, a sequence is made up of A, C, G, T, or N for an ambiguous base. Let \$ also be a special single character as a trailing empty end of the sequence S (e.g., $S = \text{"CTTGGCTGGA\$"}$). The Burrows–Wheeler transform of the sequence S or $\text{bwt}(S)$ is computed by generating cyclic rotations of the sequences, as shown in the first column of Figure 2.7. The cyclic

Cyclic rotations	Sorted rotations
CTTGGCTGGA\$	\$CTTGGCTGGA
\$CTTGGCTGGA	A\$CTTGGCTGG
A\$CTTGGCTGG	CTGGA\$CTTGG
GA\$CTTGGCTG	CTTGGCTGGA\$
GG\$CTTGGCT	GA\$CTTGGCTG
TGG\$CTTGGC	GCTGGA\$CTT
CTGGA\$CTTGG	GGA\$CTTGGCT
GCTGGA\$CTTG	GGCTGGA\$CTT
GGCTGGA\$CTT	TGG\$CTTGGC
TGGCTGGA\$CT	TGGCTGGA\$CT
TTGGCTGGA\$C	TTGGCTGGA\$C

FIGURE 2.7 Cyclic rotations and sorted rotation.

rotations then are sorted alphabetically to form a matrix called BWM (Burrows–Wheeler Matrix) as shown in the second column of Figure 2.7. The last column of characters (in red) in the BWM is the BWT of the sequence. As shown in Figure 2.8, the BWT of the sequence S is “AGG\$GGTTCTC”. When an aligner uses BWT, it transforms the entire genome sequence into a BWT. However, computing a BWT using the above naïve approach is computationally expensive with $O(n^2 \log n)$ time complexity. Instead, we can use the suffix array to compute the BWT of a reference genome in $O(n)$ time complexity. This strategy is utilized by the aligners that use BWT. Constructing a BWT from a suffix array is straightforward and very simple. To get the i th character of the BWT of the string S , whose characters are indexed as $i=1, 2, \dots, n$, from the suffix array (A), simply use the following rule:

```
BWT(S)[i] = S[A[i] - 1] if A[i] > 1 else S[n]
```

Figure 2.8 shows the indexes $i=1, 2, 3, \dots, 11$ (first row) of the sequence S (second row) and suffix array index A (third row), as shown in Figure 2.7. Now to infer the BWT from A , let us begin from $BWT(S)[11]$. By applying the rule, $BWT(S)[11]=S[A[11] - 1]=S[10]=A$. Likewise, $BWT(S)[10]=S[A[10] - 1]=S[9]=G$; $BWT(S)[6]=S[A[6] - 1]=S[5]=G$; but $BWT(S)[1]$, $A[1]$ is less than 1; hence, $BWT(S)[1]=\$$. For the $BWT(S)[i]$ corresponding to $A=9, 5, 8, 4, 7, 3, 2$, we will continue using $BWT(S)[i]=S[A[i] - 1]$. The BWT is shown in Figure 2.9.

The question that comes up is why we need to transform a sequence of a reference genome into BWT? The BWT serves two purposes; first, BWT groups the characters of the sequence so that a single character appears many times in a row because the column is sorted alphabetically and that can be used for sequence compression, which reduces the memory storage. In this sense, the BWT is compressible and reversible. The second purpose is that BWT is a data structure that can be used for indexing the sequence of a reference genome to make finding a position of a read fast.

The property of the BWT is that we can reverse it to obtain the BWM by using the last-to-first column mapping property or simply as LF mapping, where L is the last column

<i>i</i>	1	2	3	4	5	6	7	8	9	10	11
<i>S</i>	C	T	T	G	G	C	T	G	G	A	\$
<i>A</i>	11	10	6	1	9	5	8	4	7	3	2
BWT	A	G	G	\$	G	G	T	T	C	T	C

FIGURE 2.8 Constructing BWT from Suffix array.

F	L
\$	A
A	G
C	G
C	\$
G	G
G	G
G	T
G	T
T	C
T	T
T	C

FIGURE 2.9 L and F columns.

in the sorted suffixes that corresponds to the BWT and F is the first column in the sorted rotations. The L and F columns are separated as shown in Figure 2.9.

We need only the first (F) and last (L) columns to reverse the BWT to the original sorted suffixes of the string. The BWT can be recovered using the LF mapping.

Since F must begin with "\$", we know that this character is the last one in the original string. The "\$" is preceded by the first character in L, which is "A"; so now we could infer the last two characters "A\$" in the original sequence. To infer the character that comes before "A", let us find the first "A" in F, which is in the second row, so the character in the second row of L which is "G" is the character that comes before "A\$"; so now we could infer the last three characters "GA\$" of the original string. Since this "G" is the first "G" in L, it points to the first "G" in F (5th row); thus, the character that comes before "GA\$" is the character in the 5th row of L, which is "G". So now the inferred last four characters are "GGA\$". Since this "G" is the third "G" in L, it points to the third "G" in F, which is in the 7th row, and the character in the 7th row of L, which is "T", is the character that comes before "GGA\$" so the inferred last five characters are "TGGA\$". The rank of this "T" is the first "T" in L which points to the first "T" in F in the 9th row, so the character which comes before "TGGA\$" is "C" in the 9th row of L. Thus, the inferred last 6 characters of the original string are "CTGGA\$". We can continue using this L-to-F mapping until we recover the entire sequence "CTTGGCTGG\$" as indicated by the arrows in Figure 2.10. This reversion process is called backward search mechanism since it begins from the very end of the string and moves backward. A computer algorithm usually performs this task.

We can recover the original string using the same LF mapping property but in a different way. Given the LF matrix, the string S is reconstructed starting from the end of the

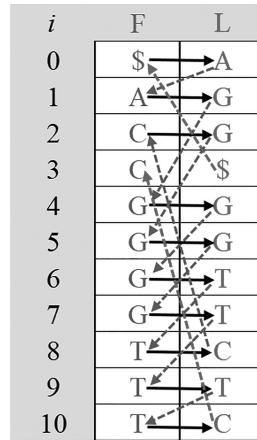


FIGURE 2.10 LF mapping to reverse the BWT.

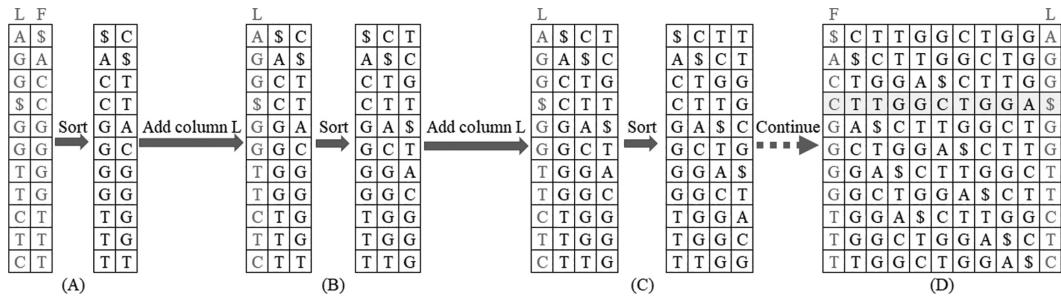


FIGURE 2.11 Using LF mapping to reverse the sorted rotations.

string, \$, which is also the first character of F column. The character before \$ will be the first character of L column. In general, the character which comes before can be inferred with the pairs L and F relationships as shown in Figure 2.10. These relationships are also illustrated in Figure 2.11 which shows that, first, we recover the first two columns of the BWM by creating a two-column matrix from columns L and F and sorting it as shown in Figure 2.11a. The third column is recovered by creating a three-column matrix from column L and the first two recovered columns and then we sort it (Figure 2.11b). The same is repeated to recover the fourth column of the sorted rotations as shown in Figure 2.11c. We will continue like this until we recover all columns and the BWM will be as shown in Figure 2.11d. The original string is the one ends with the character “\$” in the BWM as indicated in the shaded row in Figure 2.11.

The steps shown in Figures 2.10 and 2.11 are to show how a BWT is reversible. However, there are several algorithms implemented by software for reversing a BWT using LF mapping. The BWT is used as data structure for indexing a genome (compressed or uncompressed) by transforming the entire genome into a BWT. Once a genome has been transformed, there are several searching algorithms and approaches for finding the position of a substring (read) on the reference genome.

2.2.5 FM-Index

FM-index (Full-text Minute-space index) performs a backward search mechanism on the top of BWT to find exact pattern matches on a string with a linear computational time complexity $O(N)$, regardless of the length of the string. Moreover, it could achieve high compression ratios that allow the indexing of the whole human genome into less than 2.0G of memory storage [2]. An FN-index searching is performed by using LF mapping matrix. Searching for a read (a substring) with FM-index uses the *backward matching* in which the LF mapping is used repeatedly until matches for the substring are found. FM-index introduces the *rank table* and *lookup table*. A rank table is generated from the BWT (column L) using the character's rank, which is defined as the number of times a character occurs previously in the prefix $L[1\dots k]$ (i.e., lists the occurrence and order of each unique character). The rank table acts as a function $B(c, k)$, which is explained as follows: if a character c and a prefix of length k are given, then we can infer the number of times that character occurs in the prefix. A lookup table lists the index of the first occurrence of each character c from the first column (column F) of the sorted matrix. The first occurrence of the character c in the lookup table can be described as $C[c]$.

Figure 2.12 shows Burrows–Wheeler matrix (a), the rank table (b), and the lookup table (c). Using the rank table and lookup table, the LF mapping can be described as

$$LF(i) = C[L[i]] + B(L[i], i)$$

The original string can be recovered using the above rule and backward mechanisms are as follows:

The last character in the string is “\$” on the first row in column F, preceded by “A” on the first row in column K. So, the last two characters are “A\$”. Now we can use the above rule to find the character that comes before “A\$”. Since “A” is on row 1 in column L, we can locate that character in column F as $LF(1)$ as follows:

$LF(1)=C(A)+B(A, 1)=1+2=2$: The character is on row 2 in column F. So, “A” is preceded by “G”, which is on row 2 in column L. The last three characters are “GA\$”.

$LF(2)=C(G)+B(G, 2)=4+1=5$: The character is on row 5 in column F. So, “G” is preceded by “G”, which is on row 5 in column L. The last four characters are “GGA\$”.

$LF(5)=C(G)+B(G, 5)=4+3=7$: The character is on row 7 in column F. So, “G” is preceded by “T”, which is on row 7 in column L. The last four characters are “TGGA\$”.

$LF(7)=C(T)+B(T, 7)=8+1=9$: The character is on row 9 in column F. So, “T” is preceded by “C”, which is on row 9 in column L. The last four characters are “CTGGA\$”.

We can continue using that rule with the rank table and lookup table until we recover the entire string “CTTGGCTGGA\$”.

The FM-index can also be used as a pattern searching technique that operates on the BWT to map read sequences to locations on the BW-transformed reference genome. Searching for a pattern is a backward process like recovering the original string; a character is processed at a time, beginning with the last character of the pattern (read sequence) [11].

(A)

<i>i</i>	F	L								
1	\$	C	T	T	G	G	C	T	G	G
2	A	\$	C	T	T	G	G	C	T	G
3	C	T	G	G	A	\$	C	T	T	G
4	C	T	T	G	G	C	T	G	G	A
5	G	A	\$	C	T	T	G	G	C	T
6	G	C	T	G	G	A	\$	C	T	T
7	G	G	A	\$	C	T	T	G	G	C
8	G	G	C	T	G	G	A	\$	C	T
9	T	G	G	A	\$	C	T	T	G	G
10	T	G	G	C	T	G	G	A	\$	C
11	T	T	G	G	C	T	G	G	A	\$

(B)

<i>i</i>	\$	A	C	G	T
1	0	1	0	0	0
2	0	1	0	1	0
3	0	1	0	2	0
4	1	1	0	2	0
5	1	1	0	3	0
6	1	1	0	4	0
7	1	1	0	4	1
8	1	1	0	4	1
9	1	1	1	4	1
10	1	1	1	4	2
11	1	1	2	4	2

(C)

c	\$	A	C	G	T
C[c]	0	1	2	4	8

FIGURE 2.12 (a) BWT, (b) rank table, and (c) lookup table.

2.3 READ SEQUENCE ALIGNMENT AND ALIGNERS

For read mapping, we usually have millions of reads produced by a high-throughput instrument and we wish to determine the origin of each of the reads in the sequence of a reference genome. For most of sequencing applications, the read alignment or mapping is the slowest and the most computationally expensive step. This is because mapping programs will attempt to determine the most likely points of origin for each read with respect to a reference genome. Mapping the reads produced from eukaryotic RNA-Seq requires extra efforts by aligners. In eukaryote, the coding regions (exons) of the genes are separated by non-coding regions (introns). Since only transcriptome or gene transcripts are targeted in the RNA sequencing, the aligners used for mapping RNA-Seq reads must be aware of the non-contiguous nature of the exons and the challenge of the detection of the splicing regions.

Indeed, before performing alignment, we need to download the sequence of the reference genome of the species studied and then index the reference genome so that the locations, where reads maps, can be found easily upon the process of searching and alignment. For most of the aligners, indexing of the reference genome is the first step before performing read mapping. Above, we discussed the most commonly used indexing methods for storing and organizing the reference genome sequence so it can be easily searched to determine the locations (coordinates) of aligned reads. There is another challenge faced by aligners; the reads produced by sequencers may not be exactly aligned to a location in the reference genome sequence because of base call errors or may not be naturally due to mutations (substitutions, deletions, or insertions) in the DNA sequences of the individual

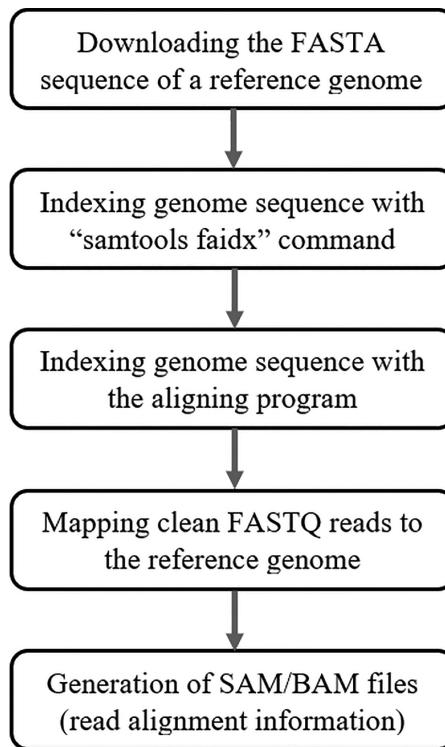


FIGURE 2.13 The general workflow of the read mapping.

studied. To overcome this challenge, aligners must adopt a strategy to perform the gapped alignment rather than performing only the exact alignment.

Read mapping is required by the most sequencing applications including reference-based genome assembly, variant discovery, gene expression, epigenetics, and metagenomics. As shown in Figure 2.13, the workflow of read mapping/alignment includes downloading the right FASTA file of the reference genome of the species studied, indexing the sequence of the reference genome with “samtools index” command, indexing the reference genome with an aligner, and finally performing mapping of the cleaned reads with the aligner itself. Remember that, before mapping, the step of quality control must be performed as discussed in Chapter 1. So, when we move to the step of read mapping, we should have already cleaned up the reads and fixed most of the failed metrics shown by the QC reports. Even if we couldn’t fix all errors, we should also be aware of the effect of that in the final results.

We have also discussed above how to download a reference genome of an organism and how to index it. To avoid repetition, we will delve into read mapping and generation of SAM/BAM files without covering the topics that we have already discussed.

Read mapping is the process of finding locations on a reference genome where reads, contained in FASTQ files, map. The read mapping information are then stored in a SAM/BAM file format, which is a special file format for storing sequences alignment

information. Most aligners are capable of performing both exact matching and inexact matching, which are essential to find the locations of reads that may have some base call errors or varied genetically from the reference genome. The different aligners implement different algorithms to perform both kinds of lookups in the indexed reference genome stored in data structures like suffix tree, suffix array, hashing table, and BWT. While the exact lookup is straightforward, the inexact matching uses sequence similarity to find the most likely locations where a read is originated. Although there are different ways to measure sequence similarity, most aligners used Hamming distance [12] or Levenshtein distance [13] to score the similarity between a reads and portions of the reference genomes based on a threshold. Some aligners use the *seed-and-extend* strategy to extend a seed (an exact matched substring) across multiple mismatched bases to allow mapping reads with base call errors or variations. Most aligners employ seed-and-extend strategy on the local sequence alignment using SW algorithm. Seeds are created by making overlapping k-mers (substrings or words of length k) from the reference genome sequence. Some aligners like Novoalign [14] and SOAP [15] index k-mers with the trie or hash table data structures for a quick search.

2.3.1 SAM and BAM File Formats

Almost all read aligning programs (aligners) store alignment information of the reads mapped to the reference genome in a Sequence Alignment and Map (SAM) file or Binary Alignment and Map (BAM) file, which is the binary form of SAM. The SAM file is a readable plain text file for storing biological sequences mainly aligned to a reference sequence [16] but it can also contain unmapped reads. It is a TAB-delimited text file consisting of two main sections: (i) a header section and (ii) an alignment section.

The header section of the SAM file is optional, and when it is present, it must be before the alignment section. Each line in the header section must start with “@” symbol followed

@HD	VN:1.0	GO:none	SO:coordinate	
@SQ	SN:1	LN:249250621	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:2	LN:243199373	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:3	LN:198022430	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:4	LN:191154276	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:5	LN:1809151260	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:6	LN:1711115067	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:7	LN:159138663	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:8	LN:146364022	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:9	LN:141213431	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:10	LN:135534747	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:11	LN:135006516	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:12	LN:133851895	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:13	LN:115169878	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:14	LN:107349540	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:15	LN:102531392	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:16	LN:90354753	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:17	LN:81195210	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:18	LN:78077248	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:19	LN:59128983	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:20	LN:63025520	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:21	LN:48129895	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:22	LN:51304566	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:X	LN:115270560	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:Y	LN:59373566	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:MT	LN:16569	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:GL000207.1	LN:4262	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/refer>
@SQ	SN:GL000226.1	LN:15008	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technic>
@SQ	SN:GL000229.1	LN:19913	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technic>
@SQ	SN:GL000231.1	LN:27386	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technic>
@SQ	SN:GL000210.1	LN:27682	AS:NCBI37	UR:ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technic>

FIGURE 2.14 A header section of a SAM file.

by one of the two-letter header record type codes. A record type code may have two-letter subtype codes. Table 2.1 lists and describes the two-letter codes of the SAM header section, and Figure 2.14 shows an example header section. Notice that the SAM file begins with “@HD VN:1.0 SO:coordinate”, which indicates that the specification of SAM version 1.0 was used and the alignments in the file are sorted by the coordinate. We can also notice that there are several @SQ header lines, each line is for the reference sequence used for the alignment. The @SQ header includes the sequence name (SN), which is the chromosome number, and sequence length (LN). The last two lines in the header section should include @PG, which describes the program used for the alignment, and @CO, which describes the command lines.

The alignment section begins after the header section. Each alignment line has 11 mandatory fields to store the essential alignment information. The alignment section may have variable number of optional fields, which are used to provide additional and aligner specific information.

Figure 2.15 shows a partial alignment section of a SAM file. The columns of the alignment section are split because they do not fit the page. Table 2.2 lists and describes 11 mandatory fields of the SAM alignment section.

These 11 mandatory fields are always present in a SAM file. If the information of any of these mandatory fields is not available, the value of that field will be replaced with “0” if its data type is integer or “*” if the data type is string. Most field names are self-explanatory.

TABLE 2.1 The Two-Letter Codes of the Header Section and Their Description

Code	Header Code Description
@HD	This header codes for metadata, and if it is present, it must be the first line of the SAM file. This header line may include subtypes: VN for format version, SO for sorting order, GO for grouping alignment, and SS for sub-sorting order of alignments
@SQ	This is for the reference sequence used for aligning the reads. A SAM file may include multiple @SQ lines for the reference sequences used. The order of the sequences defines the order of alignment sorting. The two most common sub-type codes used in this header line include SN for reference sequence name and LN for reference sequence length
@RG	This header line is used to identify read group and it is used by some downstream analysis programs (e.g., GATK) for grouping files based on the study design. Multiple lines can exist in a SAM file. This line may include the ID for the unique read group identifier, BC for the barcode sequence identifying the sample, CN for the name of sequencing facility, DS for description to be used for the read group, DT for the date of sequencing, LB for the sequencing library, PG for the programs used for processing the read group, PL for the platform of the sequencing technology used to generate the reads, PM for the platform model, PU for the platform unit, which is a unique identifier (e.g., flow cell/slide barcode), and SM for the sample identifier, which is the pool name where a pool is being sequenced
@PG	This is the header line for describing the program used to align the reads. It may include ID for the program unique record identifier, PN for the program name, CL for the command line used to run the program, PP for the previous @PG-ID, DS for description, and VN for the program version
@CO	This is the header line for a text comment. Multiple @CO lines are allowed

TABLE 2.2 Eleven Mandatory Fields of the Alignment Section of the SAM File

Column	Field	Description
1	QNAME	The query sequence name (string)
2	FLAG	Bitwise flag (integer)
3	RNAME	Reference sequence name (string)
4	POS	Mapping position from the leftmost first base (integer)
5	MAPQ	Mapping quality (integer)
6	CIGAR	CIGAR string (string)
7	RNEXT	Reference name of the mate or next read (string)
8	PNEXT	Position of the mate or next read (integer)
9	TLEN	Sequence length (integer)
10	SEQ	The read sequence (string)
11	QUAL	ASCII code of the Phred-scaled base (Phred+33) (string)

FIGURE 2.15 An alignment section of a SAM file.

The QNAME field is to show the read sequence name, which is obtained from the FASTQ file. FLAG is a bitwise integer that describes the alignments (e.g., paired, unaligned, and duplicate). The description is stored as codes as shown in Table 2.3.

The FLAG field in the SAM file may have one of the decimal values listed in Table 2.3 or the sum of any of those decimal values. For instance, Figure 2.16 shows the FLAG for the read “SRR062634.6862698” is 99, which means that this FLAG combines 4 conditions: $1+2+32+64=99$. This means that the read maps to that position of the reference genome are described as follows: the read is paired (1), the aligner mapped the two pairs properly (2), the next sequence (SEQ) is a reverse strand (32), and first read in the pair (64). Instead of doing mental math to figure out a such FLAG number, we can use “samtools flags” command as follows (Figure 2.16):

```
 samtools flags 99  
 samtools flags
```

The RNAME field shows the reference sequence name of the alignment such as a chromosome name (e.g., 1). This field will be filled with "*" for unmapped read.

```
(base) hamid@node2:~/junk$ samtools flags 99
0x63 99 PAIRED,PROPER_PAIR,MREVERSE,READ1
(base) hamid@node2:~/junk$ samtools flags
About: Convert between textual and numeric flag representation
Usage: samtools flags INT|STR[,...]

Flags:
 0x1      PAIRED      .. paired-end (or multiple-segment) sequencing technology
 0x2      PROPER_PAIR .. each segment properly aligned according to the aligner
 0x4      UNMAP       .. segment unmapped
 0x8      MUNMAP      .. next segment in the template unmapped
 0x10     REVERSE     .. SEQ is reverse complemented
 0x20     MREVERSE    .. SEQ of the next segment in the template is reversed
 0x40     READ1       .. the first segment in the template
 0x80     READ2       .. the last segment in the template
 0x100    SECONDARY   .. secondary alignment
 0x200    QCFAIL      .. not passing quality controls
 0x400    DUP         .. PCR or optical duplicate
 0x800    SUPPLEMENTARY .. supplementary alignment
```

FIGURE 2.16 Using “samtools flags” to convert numeric FLAG representations into textual.

The POS field specifies the leftmost mapping position in the reference genome of the first base of the aligned read. For the sequence “SRR062634.6862698”, the alignment position is 10001 as shown in Figure 2.15. The POS value of the unmapped read is “0”.

The MAPQ field specifies the mapping quality in Phred scoring unit. If the quality score is not available, the value will be set to 255.

The CIGAR field contains the CIGAR string, which is a sequence of the base length and the associated operations. It is used to indicate where, in the aligned read, an operation like match/mismatch, deletion, or insertion took place. Table 2.4 lists the CIGAR operations that can be associated with a read alignment. For instance, in the SAM file shown in Figure 2.15, the sequence “SRR062634.6862698” has the CIGAR string “30M70S”, which means that the first 30 bases of that sequence match (30M) bases in the reference sequence and 70 bases showing soft clip (70S), which is special mismatch due to error in the reference sequence.

The RNEXT field contains the reference sequence name of the primary alignment of the NEXT read in the template. If it is not present, the field will be set to “=” or “*”. The PNEXT field specifies the position of the primary alignment of the NEXT read in the template. This field will be set to “0” when the information is unavailable.

The TLEN field specifies the signed observed template length. This field will be positive for the leftmost read, negative for the rightmost, and undefined for any middle read.

The SEQ field contains the read sequence. This field will be set to “*” if the sequence was not stored. Otherwise, the sequence must be equal to the sum of lengths of the operations in CIGAR string. The “=” symbol indicates that the base of the read is identical to the reference base. The mapped reads are aligned to the forward genomic strand (plus strand). Therefore, if a read is mapped to the reverse strand (minus strand) of the reference sequence, the SEQ field will contain the complementary strand of the unmapped plus strand.

The QUAL field contains the Phred quality scores in ASCII character (Phred+33). The string in this field is the same as the string in the FASTQ file. This field may be set to “*” if

TABLE 2.3 The FLAG Bitwise Decimal and Hexadecimal Numbers and Their Descriptions

Decimal	Hexadecimal	Description of Read
1	0x1	The read is paired
2	0x2	The aligner mapped the two pairs properly
4	0x4	The read is unmapped
8	0x8	Next segment in the template is unmapped
16	0x10	The sequence in SEQ is a reverse strand (minus strand)
32	0x20	The next sequence (SEQ) is a reverse strand
64	0x40	First read in paired reads
128	0x80	Second read in paired reads
256	0x100	The alignment is secondary
512	0x200	The read fails platform/vendor quality checks
1024	0x400	The read is PCR or optical duplicate (technical sequence)
2048	0x800	The alignment is supplementary

TABLE 2.4 CIGAR Operations and Descriptions

Operation	Description
M	Alignment match, which can be a sequence match or mismatch
I	Insertion to the reference sequence
D	Deletion from the reference sequence
N	Skipped region from the reference sequence
S	Soft clip on the read (present in SEQ)
H	Hard clip on the read (not present in SEQ)
P	Padding (silent deletion from the padded reference sequence)
=	Sequence match
X	Sequence mismatch

the quality string is not stored. Otherwise, it must be equal to the length of the sequence in SEQ.

The alignment section of a SAM file may contain a number of optional fields. Each optional field is defined by a standard *tag* accompanied with a data type and a value in the following format:

TAG:TYPE:VALUE

The TAG is a two-character string. There are several predefined standard tags for SAM optional fields. The complete list is available at “<https://samtools.github.io/hts-specs/SAMtags.pdf>”. The user is allowed to add a new tag.

The TYPE is a single character defining the data type of the field. It can be “A” for the character data type, “B” for general array, “f” for real number, “H” for hexadecimal array, “i” for integer, and “Z” for string.

VALUE is the value of the field defined by the tag data type.

Notice that the last four columns in the SAM file shown in Figure 2.16 are for optional fields identified by the four predefined standard tags: “NH”, “HI”, “AS”, and “NM”. The “NH” tag shows the number of reported alignments (number of hits) that contain the read

sequence in the current record. The “HI” tag shows the index of the query hit. The “AS” tag shows the alignment score defined by the aligner. The “NM” tag shows the edit distance, which is defined as the minimal number of single-nucleotide edits (substitutions, insertions, and deletions) needed to transform the read sequence into the aligned segment of the reference sequence.

For more details about SAM file, read the specification of the Sequence Alignment/Map file format, which is available at “<https://samtools.github.io/hts-specs/>”.

2.3.2 Read Aligners

There are several aligners available for mapping reads to a reference genome. However, next we will discuss only BWA [9], Bowtie [17], and STAR [8] as examples. The use of other aligners is similar.

When we move to read mapping step, we will have already had cleaned the raw reads in FASTQ files as discussed in Chapter 1. In the following sections, we will show you how to map reads contained in FASTQ files to a reference genome. For this purpose, we will download FASTQ files (run # is SRR769545) from the NCBI SRA database. To avoid repeating the quality control steps, we will assume that we have cleaned the FASTQ files following the steps in Chapter 1 and the files are ready for mapping. Our example raw data are paired-end reads from the 1000 Genomes whole exome sequencing of an individual from the Great Britain population. We can download the two FASTQ files (forward and reverse) using the SRA-toolkit. The following commands create the directory “data” where the two FASTQ files will be downloaded:

```
mkdir data
cd data
fasterq-dump --verbose SRR769545
```

The size of each file is 11G; the two files take around 22G of storage space. To save storage space, we can compress these files using gzip utility, which will reduce each file to only 2.6G. Most aligners accept the gzipped FASTQ files.

```
gzip SRR769545_1.fastq
gzip SRR769545_2.fastq
```

The “.gz” will be added to the name of each file to indicate that the two files were compressed with gzip.

We can also run FastQC and display the QC reports as follows:

```
fastqc SRR769545_1.fastq.gz SRR769545_2.fastq.gz
firefox SRR769545_1_fastqc.html SRR769545_2_fastqc.html
```

The per base quality reports for the two FASTQ files are shown in Figure 2.17. We can notice that the reads in the two files have a good quality.

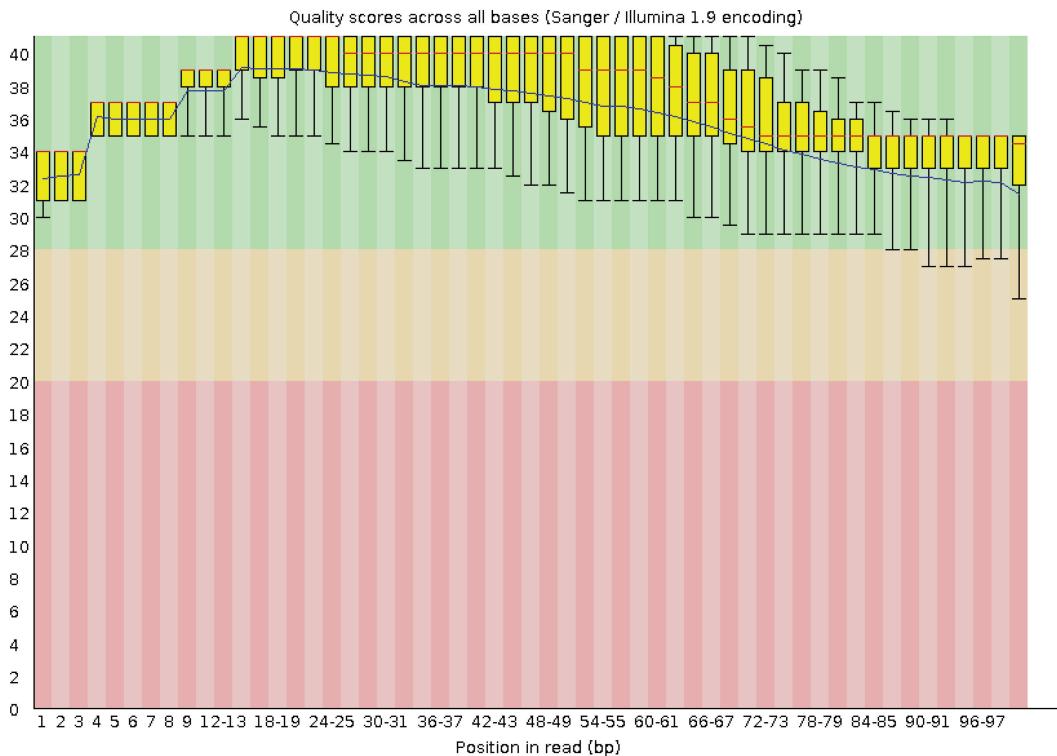


FIGURE 2.17 The per base quality reports for the reads in the two FASTQ files.

In Section 2.1, we showed how to download the FASTA file of the reference genome sequence of an organism and how to index it using “samtools faidx”. So, if you did not do that, follow the steps in that section to download the human reference and then to index it. The sequences of reference genomes can also be downloaded from other databases such as UCSC database. We have also downloaded and compressed example paired-end FASTQ files for practice. The next step is to show you how to use an aligner (BWA, Bowtie, and STAR) for read mapping.

2.3.2.1 Burrows–Wheeler Aligner

The Burrows–Wheeler Aligner (BWA) is a sequence aligner that uses BWT and FM-index. We can install the latest version of the BWA software by following the installation instructions at “<https://github.com/lh3/bwa>” or we can use the following commands:

```
git clone https://github.com/lh3/bwa.git
cd bwa; make
```

The above will clone the BWA source files into your working directory and then it will compile it. Once BWA has been installed successfully, you may need to set its path so that

you will be able to use it from any directory. While you are in the “bwa” directory run the “pwd” command to print the absolute path of BWA, copy it, then change to your home directory, and open “.bashrc” file using “vim” or any text editor of your choice:

```
cd $HOME
vim .bashrc
```

Add the following to the end of the “.bashrc” file:

```
export PATH="your_path/bwa":$PATH
```

Do not forget to replace “your_path” with the path to the “bwa” directory on your computer. Save the “.bashrc” file, exit, and restart the terminal for the change to take effect. Type “bwa” on the terminal and press the enter key. If the BWA software was installed and added to the path correctly, you will see the help screen.

BWA has three alignment algorithms: BWA-MEM “bwa mem”, BWA-SW “bwa bwasw”, and BWA-backtrack “bwa aln/samse/sampe”. Both “bwa mem” and “bwa bwasw” algorithms are used for mapping short and long sequences produced by any of the sequencing technologies. The “bwa aln/samse/sampe” also called BWA-backtrack is designed for Illumina short-sequence reads up to 100 bp. Among the three algorithms, “bwa mem” is the most accurate and the fastest.

Indeed, aligning read sequences to a reference genome with BWA requires indexing the reference genome using “bwa index” command. We can use this command to index the human reference genome which was downloaded and indexed with “samtools faidx” above as follows:

```
bwa index GRCh38.p13_ref.fna
```

The indexing will take some time depending on the size of the reference genome and the memory of your computer. When the “bwa index” command finishes indexing, it will display the information, including the number of iterations, the elapsed time in second, the indexed FASTA file name, and the real time and CPU time taken for the indexing process. The indexing of the human genome may take up to six hours on a desktop computer of 32G RAM.

The BWA indexing process creates five bwa index files with extensions “.amb”, “.ann”, “.bwt”, “.pac”, and “.sa”. The total storage space for the current human reference genome and its index files is around 9.4G.

The “.amb” file indexes the locations of the ambiguous (unknown) bases in the FASTA reference file that are flagged as N or another character but not as A,C,G, or T. The “.ann” file contains annotation information such as sequence IDs and chromosome numbers. The “.bwt” is a binary file for the Burrows–Wheeler transformed sequence. The “.pac” is a binary file for the packed reference sequence. The “.sa” is also a binary file containing the suffix array index. For mapping read sequences to the reference genome, all these five files must be together in the same directory.

So far, we have two directories: “data” where the FASTQ files were downloaded and “refgenome” where the FASTA sequence of the reference genome was downloaded and indexed.

In the next step, we will use any of the alignment algorithms to map the reads to the human reference genome. The “bwa mem” algorithm can be tried first as recommended by the BWA.

1-BWA-MEM algorithm

In the following, the “bwa mem” command maps the reads in the FASTQ files to the human reference genome and saves the output file, which contains the mapped reads in a new directory called “bwa_sam”. Before running the commands, make sure that the working directory is just one level out of the two directories.

```
mkdir sam
bwa mem \
-t 4 \
refgenome/GRCh38.p13_ref.fna \
data/SRR769545_1.fastq.gz \
data/SRR769545_2.fastq.gz \
> sam/SRR769545_mem.sam 2> sam/SRR769545_mem.log
```

The “bwa mem” command has the capability of mapping reads of a length ranging between 70 bp and 1 Mbp. The BWA-MEM algorithm uses seeding alignments with maximal exact matches (MEMs) and then it extends seeds with the affine-gap SW algorithm. We can run “bwa mem” on the command line without any option to learn more about “bwa mem” command. In the above, we use “-t 4” so that the program can use four parallel threads to perform read mapping. Then, we provided the path of the FASTA file name of the reference genome and the two FASTQ file names. The output of the aligned reads will be redirected to a new file “SRR769545_mem.sam” using the Unix/Linux redirection symbol “>”. The command execution comments are redirected to the log file “SRR769545_mem.log” using “2>”, which is used in Unix/Linux to redirect stderr (standard error) to a text file. Both the output files will be saved in the “sam” directory. The log file contains important information about the comments and any standard error that occurs during the mapping process. Always open the log file if the running fails. The log file will tell you if the failure is due to the wrong syntax, wrong file path, or any other standard error. Even if the program has finished running normally, it is a good practice to open the log file and look at the statistics and comments.

```
cd sam
less SRR769545_mem.log
```

You can also display the SAM file produced by “bwa mem” with the following:

```
less -S SRR769545_mem.sam
```

The size of this SAM file is about 19G. We can convert it to the BAM format using Samtools to save some storage space. First, you need to change to the “sam” directory and then run the following:

```
samtools view \
-uS \
-o SRR769545_mem.bam \
SRR769545_mem.sam
```

The new BAM file is about 15G. We can then delete the SAM file as follows to save some storage space:

```
rm SRR769545_mem.sam
```

BWA-MEM2 is an optimized BWA-MEM algorithm that has been recently released. This new version produces alignments identical to BWA-MEM but it is faster and the indexing occupies less storage space and memory [18]. You can install BWA-MEM2 separately by following the instructions available at “<https://github.com/bwa-mem2/bwa-mem2>”.

2-BWA-SW

The BWA-SW [9] algorithm, like BWA-MEM, can also be used for the alignment of single- and paired-end long reads generated by all platforms. It uses SW local alignment approach to map reads to a reference genome. BWA-SW has a better sensitivity when alignments have frequent gaps. However, this algorithm has been depreciated by his developer since BWA-MEM is restructured for better performance. The following “bwa bwasw” performs read alignment as above:

```
bwa bwasw \
-t 4 \
refgenome/GRCh38.p13_ref.fna \
data/SRR769545_1.fastq.gz \
data/SRR769545_2.fastq.gz \
> sam/SRR769545_bwasw.sam 2> sam/SRR769545_bwasw.log
```

You can convert this SAM file to BAM file as we did above or you can just delete it to save some space.

3-BWA-backtrack

The BWA-backtrack algorithm is designed for aligning Illumina short reads of a length up to 100 bp with sequencing error rates below 2%. It involves two steps: (i) using “bwa aln” to find the coordinates of the positions, where the short reads align, on the reference genome, and then (ii) generating alignments with “bwa samse” for single-end reads or “bwa sampe” for paired-end reads. The base call quality usually deteriorates toward the end of reads generated by Illumina instruments. This algorithm optionally trims low-quality bases from the 3'-end of the short reads before alignment. Therefore, it is able to align more reads with high error rate toward the 3'-ends of the reads.

In the following, we will use “bwa aln” to perform the first step of the alignment. Run “bwa aln” without any option on the command line to learn more about the usage and options. If the quality of reads at the 3'-end is low, we can use the “-q” option with this command to specify a quality threshold for read trimming down to 35 bp. Run the following commands while you are one step out of the “refgenome” and “data” directories:

```
bwa aln \
  refgenome/GRCh38.p13_ref.fna \
  data/SRR769545_1.fastq.gz \
> data/SRR769545_1.sai

bwa aln \
  refgenome/GRCh38.p13_ref.fna \
  data/SRR769545_2.fastq.gz \
> data/SRR769545_2.sai
```

Then, we can use “bwa sampe” to generate the SAM file for the alignments.

```
$ bwa sampe \
  refgenome/GRCh38.p13_ref.fna \
  data/SRR769545_?.sai \
  data/SRR769545_?.fastq.gz \
> sam/SRR769545_aln.sam 2> sam/SRR769545_aln.log
```

2.3.2.2 Bowtie2

Bowtie2 is an aligner that uses BWT and FM-index as data structures for indexing the reference genome. It is an ultrafast, memory-efficient short read aligner, and it allows mapping millions of reads to a reference genome on a typical desktop computer. Bowtie2 is the next generation of the original Bowtie which requires the reads to have equal length and it does not align reads with gaps. Bowtie2 was developed to overcome those limitations. It performs read mapping in four steps: (i) extraction of seeds from the reads and their reverse strands, (ii) using FM-index for exact ungapped alignment of the seeds, (iii) sorting the alignments by scores and identifying the alignment position on the reference genome from the index, and (iv) extending seeds into full alignments using parallel dynamic programming [16]. Bowtie2 can be installed on Linux with the following commands:

```
git clone https://github.com/BenLangmead/bowtie2.git
cd bowtie; make
```

Then, you need to set Bowtie2 path so that you can run it from any directory by editing “.bashrc” file from your home directory.

```
cd #HOME
vim .bashrc
```

Add the following to the end of the file:

```
export PATH="your_path/bowtie2":$PATH
```

Do not forget to change “your_path” with the right path on your computer. Save the file and exit. You may need to restart the terminal or run “source .bashrc” to make the change active. Then, you can enter “bowtie2” on the terminal. If Bowtie2 is installed and its path was set, help screen will be displayed.

Before read mapping, we need to use “bowtie2-build” command to index the FASTA sequence of the reference genome. Enter “bowtie2-build” on the command line of the terminal to display the help screen that shows the usage and options. The general syntax is as follows:

```
bowtie2-build [options] <reference_in> <ebwt_outfile_base>
```

The “bowtie2-build” command requires a FASTA file of a reference genome as an input and a prefix string which is added as a prefix to the file names of the index. The following command indexes the human genome for Bowtie2. Before running the command, make sure that the current working directory is a one-level out “refgenome” directory, where we downloaded the human genome.

```
bowtie2-build \
--threads 4 \
refgenome/GRCh38.p13_ref.fna \
refgenome/bowtie2
```

The indexing may take around 25 minutes using four processors on a computer with 32G of memory. The “bowtie2-build” command generates six index files prefixed with the prefix string provided for the command. Pre-built indexes for some organisms can also be downloaded from the official Bowtie2 website.

After indexing the reference genome, we can use “bowtie2” command to align the paired-end reads and to generate SAM file:

```
bowtie2 -x refgenome/bowtie2 \
-1 data/SRR769545_1.fastq.gz \
-2 data/SRR769545_2.fastq.gz \
-S sam/SRR769545_bowtie2.sam
```

Instead of “-S” option to generate a SAM file, we can use “-b” option to generate a BAM file. To learn more about Bowtie2’s options, enter “bowtie2” on the command line of the Linux terminal.

2.3.2.3 STAR

STAR, which stands for Spliced Transcripts Alignment to a Reference, is a fast read aligner developed to handle the alignment of massive number of RNA-Seq reads. Its alignment

algorithm uses uncompressed suffix array (SA) data structure to perform sequential maximum mappable seed search, which is defined by the developer as the longest substring of a read that matches exactly one or more substrings of the reference genome. This search is achieved by mapping seeds to the reference genome. A read with a splice junction site will not be mapped continuously. The algorithm will try to align the first unmapped seed to a donor splice site and then it repeats the search and aligns the unmapped to an acceptor splice site. The search is performed for forward and reverse direction. This kind of search will help in the detection of base mismatches and InDels. If a single or multiple mismatches are found, the matched substrings will act as anchors on the genome to allow extension. The search is then followed by a seed clustering by proximity for determining the anchor seeds. Then, the aligned seeds around the anchor seeds within a user-defined window are stitched together using dynamic programming. STAR is capable of detecting splices and chimeric transcripts and mapping complete RNA transcripts that are formed from non-contiguous exons in eukaryotes [8].

The STAR software can be installed by following the installation instructions, which are available at “<https://github.com/alexdobin/STAR>”. On Ubuntu, you can install STAR using the following command:

```
sudo apt install rna-star
```

As most of the read aligners, STAR basic workflow includes both index generation and read alignment. However, for index generation, both a reference genome in the FASTA format and reference annotation file in GTF format are required. Pre-built indexes for genomes of some species can be downloaded from the STAR official website. As discussed before, the reference genomes can be downloaded from databases such as NCBI Assembly, UCSC genome collection, or any other database. For the aligners discussed before, we downloaded the human reference genome from the NCBI Genome database. For STAR, we will download the human reference genome and its GTF annotation file from the UCSC database. The reason is that UCSC maintains the gene annotation file in GTF format. Use the following command to create a new directory “ucscref” and then download and decompress the human reference genome and GTF annotation file:

```
mkdir ucscref
wget \
-O "ucscref/hg38.fa.gz" \
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/latest/
hg38.fa.gz
wget \
-O "ucscref/hg38.fa.gz" \
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/genes/
hg38.ncbiRefSeq.gtf.gz
gzip -d ucscref/hg38.fa.gz
gzip -d ucscref/hg38.ncbiRefSeq.gtf.gz
```

Then, we will build the index for the reference genome using the “STAR” command.

```
mkdir indexdir
STAR --runThreadN 4 \
  --runMode genomeGenerate \
  --genomeDir indexdir \
  --genomeFastaFiles ucscref/hg38.fa \
  --sjdbGTFfile ucscref/hg38.ncbiRefSeq.gtf \
  --sjdbOverhang 100
```

Above, with the “STAR” command, we used “--runThreadN” to specify the number of threads used for indexing, “--runMode genomeGenerate” to tell the command that we wish to generate a genome index, “--genomeDir” to specify the directory where the index files are to be saved, “--genomeFastaFiles” to specify the file path of the reference genome FASTA file, “--sjdbGTFfile” to specify the file path of the annotation GTF file, and “--sjdbOverhang” to specify the length of the genomic read around the annotated junction to be used in constructing the splice junctions database. For this option, we can provide read size minus one (n-1) if the read size is equal for all reads; otherwise, we can provide the maximum size minus one.

The process of indexing may take a long time and may consume much memory and storage space compared to the other aligners. Several files will be generated including binary genome sequence files, files of the suffix arrays, a text file for the chromosome names or lengths, splice junctions’ coordinates, and transcripts/genes information. Those files are for the STAR internal use; however, the chromosome names can be renamed in the chromosome file if needed.

The next step is to use STAR command for aligning the reads. This time we will use “--runMode alignReads” to tell the program to run read mapping mode, “outSAMtype BAM Unsorted” to generate an unsorted BAM file, “--readFilesCommand zcat” to tell the program that the FASTQ files are compressed, “--genomeDir” to specify the index directory, “--outFileNamePrefix” to specify the prefix for the output files, and “--readFilesIn” to specify the FASTQ file names. You can also set “outSAMtype BAM SortedByCoordinate” to generate a BAM file sorted by the alignment coordinates. However, that will exhaust the memory of a 32G-RAM computer.

```
STAR --runThreadN 4 \
  --runMode alignReads \
  --outSAMtype BAM Unsorted \
  --readFilesCommand zcat \
  --genomeDir indexdir \
  --outFileNamePrefix STARoutput/SRR769545 \
  --readFilesIn data/SRR769545_1.fastq.gz data/SRR769545_2.
  fastq.gz
```

STAR alignment mode produces a BAM file, containing read alignment information, and four text files, three log files with file names “*Log.out”, “*Log.progres.out”, and “*Log.final.out”, where “*” is for the prefix specified by “--outFileNamePrefix” option,

and a tab-delimited file with an extension “*SJ.out.tab”. The “*Log.out” file contains detailed information about the run. The “*Log.progress.out” file contains statistics for the alignment progress in 1-minute intervals. The “*Log.final.out” file contains important summary statistics for the alignment after the alignment is complete. We will discuss this later in the RNA-Seq chapter. “Log.final.out” contains information about the splice junctions.

2.4 MANIPULATING ALIGNMENTS IN SAM/BAM FILES

Before proceeding to the next step of the analysis workflow, we may need to manipulate the alignments in the SAM/BAM file. There are several programs used to serve that purpose but Samtools and PICARD are the most commonly used ones. In the following, we will discuss how to use Samtools to manipulate alignments in SAM/BAM files. PICARD will be discussed in the RNA-Seq chapter.

2.4.1 Samtools

Samtools is a collection of command-line utilities for the manipulation of alignments in the SAM/BAM files. These utilities are grouped into indexing, editing, file operations, statistics, and viewing tools. Samtools can be installed on Linux by running “sudo apt-get install samtools” or you can follow the installation instructions available at “<http://www.htslib.org/download/>”. After installation, you can display the complete list of the Samtools’ utilities by running “samtools” without any option on the command line.

A Samtools utility is executed using the following format:

```
 samtools <command> [options]
```

where <command> is any Samtools command and [options] is any of the command options.

The most used commands include “cat”, “index”, “markup”, “rmdup”, “sort”, “mpileup”, “coverage”, “depth”, “flagstats”, and “view”.

To learn more about the usage and options of any of these commands, you can use the following format:

```
 samtools <command>
```

For instance, to display the usage and options of “index” command, run “samtools index”.

In the following, we will discuss the most common uses of Samtools utilities.

2.4.1.1 SAM/BAM Format Conversion

The “samtools view” command can be used to convert SAM file into BAM file and vice versa. If you did not delete the SAM files, which were produced above and saved in “sam” directory, you can use any one of them for this practice.

From inside “sam” directory, for instance, you can run the following command to convert “SRR769545_mem.sam” to a BAM file “SRR769545_mem.bam”:

```
samtools view \
-@ 4 \
-uS -o SRR769545_mem.bam SRR769545_mem.sam
```

The “-@” option specifies the number of threads, the “-u” option is to produce uncompressed BAM output, “-S” is to ignore auto-detection of the input format, and “-o” specifies the output file.

If you have already run the above command and the old SAM file is still there, you can delete it with “rm SRR769545_mem.sam” command and then run the following to convert the BAM file back to a SAM file:

```
samtools view \
-@ 4 -h \
-o SRR769545_mem.sam \
SRR769545_mem.bam
```

The “-h” option is to include header in SAM output.

2.4.1.2 Sorting Alignments

Sorted alignments are required for some applications such as variant calling and RNA-Seq. Samtools can sort the alignments in a BAM file by coordinate order, by read name, or by a TAG. By default, “samtools sort” will sort the alignment by the coordinate order, unless “-n” or “-t TAG” option is used. If “-n” option is used, the command will sort the alignments by read name. If the “-t TAG” option is used, then the alignments will be sorted by tag. Refer to SAM and BAM file formats discussed above to learn about the standard TAGs. In the following, the BAM file will be sorted by coordinate order:

```
samtools sort \
-@ 4 \
-T mem.tmp.sort \
-o SRR769545_mem_sorted.bam \
SRR769545_mem.bam
```

This will create a new BAM file with sorted alignments. You can delete the unsorted BAM file if you need to save some storage space.

2.4.1.3 Indexing BAM File

Before using a BAM file in any of the downstream analysis, it can be indexed to allow fast random access to the alignments in the file. Thus, the alignment information will be processed faster. We can use the “samtools index” command to index the above sorted BAM file as follows:

```
samtools index SRR769545_mem_sorted.bam
```

This will generate the BAM index file “SRR769545_mem_sorted.bam.bai”.

2.4.1.4 Extracting Alignments of a Chromosome

Sometimes, we may need to work with alignments of a specific chromosome or a specific region of the genome. With the following command, you can split the alignments of a chromosome 11 in a separate file using “samtools view”:

```
samtools view SRR769545_mem_sorted.bam NC_000011.10 > chr11_human.sam
```

You can use any of the reference sequence names in the RNAME field of the SAM/BAM file, so you may need to display the content of the file to check how the reference sequences/chromosomes are named.

2.4.1.5 Filtering and Counting Alignment in SAM/BAM Files

To filter alignments in a SAM/BAM file, we can use “samtools view” with “grep” which is a Linux command for searching plain-text datasets for lines that match a regular expression. For instance, to search for the alignments with chimeric reads, which are tagged as “SA:” an optional SAM/BAM field, we can use the following:

```
samtools view SRR769545_mem_sorted.bam | grep 'SA:' | less -S
```

The chimeric read is the one that aligns to two distinct portions of the genome with little or no overlap.

To count the number of chimeric reads, we can use “wc -l” command.

```
samtools view SRR769545_mem_sorted.bam | grep 'SA:' | wc -l
```

We can also use the option “-c” with “samtools view” to count the number of reads in a BAM file:

```
samtools view -c SRR769545_mem_sorted.bam
```

We can use values in FLAG field of the SAM/BAM file to count the number of reads defined by a specific FLAG value. For instance, since the unmapped reads will be flagged as “0x4” in BAM files, we can count all mapped reads by excluding the unmapped from counting using the “-F” option.

```
samtools view -c -F 0x4 SRR769545_mem_sorted.bam
```

To count unmapped reads, use the “-f” option instead of the “-F” option as:

```
samtools view -c -f 0x4 SRR769545_mem_sorted.bam
```

We can also use the “samtools view” command together with some Unix/Linux commands and pipe symbol “|” to perform more complex count. For instance, we can count

the number of insertions (I) or deletion (D) from the CIGAR strings in a BAM file. Since the CIGAR field is the sixth column, first, we will use “samtools view -F 0x4 SRR769545_mem_sorted.bam” to extract the mapped records. Then, we can transfer that output to “cut -f 6” using the pipe symbol “|” to separate the sixth column. The output is then transferred to “grep -P” to select only the strings that have either the character “D” or “I” using the class pattern “[ID]” to match any of the two characters. Then, the output is transferred to the “tr” command to delete any characters other than “I” and “D”. Finally, the output is transferred to the “wc -c” command to count the remaining characters:

```
samtools view \
-F 0x4 SRR769545_mem_sorted.bam \
| cut -f 6 \
| grep -P '[ID]' \
| tr -cd '[ID]' \
| wc -c
```

To count insertions and deletions separately, use the following, respectively:

```
samtools view \
-F 0x4 SRR769545_mem_sorted.bam \
| cut -f 6 \
| grep -P 'I' \
| tr -cd 'I' \
| wc -c
samtools view \
-F 0x4 SRR769545_mem_sorted.bam \
| cut -f 6 \
| grep -P 'D' \
| tr -cd 'D' \
| wc -c
```

Refer to Table 2.3 for the different FLAG values and descriptions.

2.4.1.6 Removing Duplicate Reads

Duplicate reads may be produced from the library construction, PCR amplification (PCR duplicates), or a fault in the sequencing optical sensor (optical duplicates). A large number of duplicate reads originating from a single fragment may create a bias in some applications, such as RNA-Seq, in which the count of reads has a biological interpretation. The “samtools rmdup” command can be used to remove potential duplicate reads from BAM/SAM files. If multiple reads have identical coordinates, only the read (read pair if paired end) with the highest mapping quality will be retained. By default, this command works for paired-end reads. The option “-s” is used if the reads are single end.

```
samtools rmdup \
SRR769545_mem_sorted.bam \
```

```
SRR769545_mem_rmdup.bam 2> SRR769545_mem_rmdup.log
```

The above command removes the duplicate reads from the BAM file (paired end). If we need paired-end reads to be treated as single end, use “-S” option.

2.4.1.7 Descriptive Statistics

Some Samtools utilities including “flagstat”, “coverage”, and “depth” can provide simple statistics on a BAM file.

```
samtools flagstat SRR769545_mem_sorted.bam
samtools coverage SRR769545_mem_sorted.bam > coverage.txt
samtools depth SRR769545_mem_sorted.bam > depth.txt
```

For other Samtools commands, you can check the Samtools documentation which is available at “<http://www.htslib.org/doc/samtools.html>”.

2.5 REFERENCE-GUIDED GENOME ASSEMBLY

The reference-guided genome assembly is the use of a reference genome of an organism as a guide to assemble a new genome. This kind of assembly is used when a genome is re-sequenced to obtain better quality genome assembly or for variant discovery and haplotype construction. The reference-guided genome assembly is widely used to sequence the genome of individuals of the same species as of the reference genome to detect the genotypes that may associate with certain phenotype or diseases such as cancers, viral and bacterial variants or strains. It can also be used to assemble the genomes of closely related species who do not have available reference genomes. Sequences of the whole genome of an individual are used in the assembly. The workflow is the same as shown in Figure 2.13 until the point of creating SAM/BAM file. The additional step is that the aligned reads are piled up to create consensus sequences from the overlapped contiguous aligned reads. These consensus sequences are called contigs. From these contigs, only the different bases (variants) are used to edit the sequence of the reference genome to create a new genome sequence.

For the following practice, you can use any of the SAM files produced by BWA or Bowtie2 above or you can run the following commands to download the FASTQ file from the NCBI SRA database and decompress them, to download the human reference genome from UCSC database and index it, and then to perform read mapping with Bowtie2 to produce a SAM file:

```
mkdir ref_guided_ass
cd ref_guided_ass
mkdir data
fasterq-dump --verbose SRR769545
gzip SRR769545_1.fastq
gzip SRR769545_2.fastq
cd ..
mkdir ref
```

```

cd ref
wget https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.
fa.gz
gunzip -d hg38.fa.gz
samtools faidx hg38.fa
bowtie2-build --thread 4 hg38.fa hg
cd ..
mkdir sam
bowtie2 -x \
    ref/hg \
    -1 data/SRR769545_1.fastq.gz \
    -2 data/SRR769545_2.fastq.gz \
    -S sam/SRR769545.sam

```

This time we have chosen to download the UCSC human reference genomes because the chromosome names are given instead of accession numbers.

Once the SAM file has been created, we can convert it into a BAM file, sort it, and index it using Samtools.

```

samtools view -uS -o sam/SRR769545.bam sam/SRR769545.sam
samtools sort -@ 4 -T bowtie2.tmp.sort \
    -o sam/SRR769545_sorted.bam sam/SRR769545.bam
samtools index sam/SRR769545_sorted.bam

```

Then, we can use bcftools to detect the variants like variant substitutions and write that information into a binary variant call format (BCF), which is the binary form of variant call format (VCF). This file includes the difference in genotype between the reference genome and the genome of the individual sequenced. The variant discovery and VCF file format will be discussed in detail in Chapter 4. However, they have to be used here to show how the reference-guided genome assembly is performed. The bcftools can be installed in Linux as follows:

```
sudo apt install bcftools
```

Now, we can use the “bcftools mpileup” command to collapse the pileup of the reads aligned to the reference genomes in the BAM file and then to write only the variant information into a VCF file. In the following, the variant information is written into a BCF file and then that binary file is converted into a VCF file:

```

bcftools mpileup -f ref/hg38.fa \
    sam/SRR769545_sorted.bam \
    | bcftools call -mv -Ob \
    -o sam/SRR769545.bcf
bcftools convert -O v \
    -o SRR769545.vcf \
    SRR769545.bcf

```

The VCF can be compressed using the Linux compression program “bgzip” and then the compressed file is indexed using tabix program, which is a tool for indexing large bioinformatics text files. The tabix program can be installed on Linux using:

```
sudo apt install tabix
```

The following commands compress the VCF file and index it:

```
bgzip -c SRR769545.vcf > SRR769545.vcf.gz
tabix -p vcf SRR769545.vcf.gz
```

The final step in the reference-guided genome assembly is to create a consensus sequence by transferring the sequence reference genome, which was used to create the original BAM file, to the “bcftools consensus” command that utilizes the indexed variant call file to create a new genome sequence for the individual studied.

```
cat ../ref/hg38.fa \
| bcftools consensus SRR769545.vcf.gz \
> SRR769545_genome.fasta
```

The “SRR769545_genome.fasta” is the FASTA genome sequence that incorporates variants genotyped for the individual from whom the whole genome was sequenced. If a region of the reference genome is uncovered by the reads, this assembly method may skip some variants; therefore, high sequence coverage is required. Instead of incorporating the uncovered regions of the reference genome in the new sequence, the bases of those regions can be masked by Ns. There are several more advanced programs for reference-guided genome such as RATT [19].

There is a different approach to sequence a genome of an organism from scratch without using a reference genome. This approach is called de novo assembly which is discussed in Chapter 3.

2.6 SUMMARY

Except for the sequencing applications that use de novo genome assembly, read mapping to a reference genome is the most fundamental step in the workflow of the sequencing data analysis. In the NGS or TGS, the DNA molecules are fragmented into pieces in the library preparation step and then the DNA libraries are sequenced to produce the raw data in the form of millions of reads of specific lengths. The lengths of the reads produced by a high-throughput sequencing instrument vary based on the technology used into short reads (50–400 bp) or long reads (>400 bp). The quality control step to assess and preprocess the raw reads is essential to reduce the errors in the base calling and the biases that may arise due to the presence of technical reads. Indeed, read mapping is the most computationally expensive step in the sequencing data analysis workflow. That is because the alignment program attempts to determine the points of origin for millions or billions of reads in a reference genome. The alignment requires even more efforts for RNA-Seq read

mapping because the aligner should also be able to detect the splice junctions. The process of alignment is usually complicated by the possible existence of mismatches, which may be due to base call errors or due to genetic variations in the individual genome. In general, aligners are required to use a strategy that enables them to perform both an exact search and an inexact search to allow locating positions of reads with mismatches. Almost all read aligners perform alignment in two major steps: indexing of the sequence of the reference genome and finding the most likely locations of the reads in the reference genome. The FASTA sequence of the reference of an organism can be downloaded from genome databases such as NCBI Genome and UCSC database. The FASTA genome sequence is indexed first by the “samtools faidx” command to allow fast processing by the aligners. The commonly used data structures for genome indexing include BWT, FM-index, suffix arrays, and hash table for their memory efficiency and capability to store a genome sequence. There are a variety of aligners that use different indexing and lookup algorithms. We discussed only BWA, Bowtie2, and STAR. However, those are only examples. Before using an aligner, you may need to know its memory efficiency and whether it is capable to use short reads, long reads, or both. If you have RNA-Seq reads, you may also need to know whether that aligner is capable to detect splice junctions or not. Both BWA and Bowtie2 are general purpose aligners that can be used for all kinds of reads and they can operate well on a desktop computer with 32GB of RAM or more. STAR is better for RNA-Seq read and it can also run on a desktop computer but it requires much more memory for both indexing and mapping.

Almost all aligners produce SAM/BAM files, which store read mapping information. A SAM/BAM file consists of a header section and an alignment section. The alignment section includes nine mandatory columns; each row of the columns contains the mapping information of a read. The alignment information includes read name, FLAG, reference sequence name (e.g., chromosome name or accession), position in the reference sequence (coordinate), mapping quality, CIGAR string, reference name of the mate, position of the mate, read length, segment of the read sequence, and Phred base quality. FLAG field stores standard codes that describe the alignment (e.g., unmapped reads, duplicate reads, and chimeric alignments). The CIGAR string describes the operations that took place on the reads such as matches, mismatches, insertions, and deletions.

SAM/BAM files can be manipulated by some programs like Samtools and PICARD. The file manipulation includes format conversion, indexing, sorting, displaying, statistics, viewing, and filtering.

The SAM/BAM files are used in the downstream data analysis such as reference-guided genome assembly, variant discovery, gene expression (RNA-Seq data analysis), epigenetics (ChIP-Seq data analysis), and metagenomics as we will discuss in coming chapters.

REFERENCES

1. Needleman SB, Wunsch CD: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* 1970, 48(3):443–453.
2. Chacón A, Moure JC, Espinosa A, Hernández P: n-step FM-index for faster pattern matching. *Proc Comput Sci* 2013, 18:70–79.

3. Smith TF, Waterman MS: Identification of common molecular subsequences. *J Mol Biol* 1981, 147(1):195–197.
4. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: Basic local alignment search tool. *J Mol Biol* 1990, 215(3):403–410.
5. Fredkin E: Trie memory. *Commun ACM* 1960, 3:490–499.
6. Ukkonen E: On-line construction of suffix trees. *Algorithmica* 1995, 14(3):249–260.
7. Shrestha AMS, Frith MC, Horton P: A bioinformatician's guide to the forefront of suffix array construction algorithms. *Brief Bioinfor* 2014, 15(2):138–154.
8. Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M, Gingeras TR: STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* 2013, 29(1):15–21.
9. Li H, Durbin R: Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics* 2009, 25(14):1754–1760.
10. Langmead B, Trapnell C, Pop M, Salzberg SL: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* 2009, 10(3):R25.
11. Fernandez E, Najjar W, Lonardi S: String Matching in Hardware Using the FM-Index. In: *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*: 1–3 May 2011 2011. 218–225.
12. Sankoff D, Kruskal J, Nerbonne J: *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*: Cambridge University Press; 2000.
13. Levin LA: Problems of Information Transmission. In: 1973.
14. Mu JC, Jiang H, Kiani A, Mohiyuddin M, Bani Asadi N, Wong WH: Fast and accurate read alignment for resequencing. *Bioinformatics* 2012, 28(18):2366–2373.
15. Li R, Li Y, Kristiansen K, Wang J: SOAP: short oligonucleotide alignment program. *Bioinformatics* 2008, 24(5):713–714.
16. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R: The sequence alignment/map format and SAMtools. *Bioinformatics* 2009, 25(16):2078–2079.
17. Langmead B, Salzberg SL: Fast gapped-read alignment with Bowtie 2. *Nature Methods* 2012, 9(4):357–359.
18. Vasimuddin M, Misra S, Li H, Aluru S: Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*: 20–24 May 2019 2019. 314–324.
19. Otto TD, Dillon GP, Degrave WS, Berriman M: RATT: Rapid Annotation Transfer Tool. *Nucleic Acids Res* 2011, 39(9):e57.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

De Novo Genome Assembly

3.1 INTRODUCTION TO DE NOVO GENOME ASSEMBLY

In the previous chapter, we discussed reads mapping to the reference genomes of organisms which have available reference genome sequences and we also discussed the reference-based genome assembly. So, what if there is no reference genome available for an organism or we need to sequence a genome of an unknown organism. In this case, aligning reads to a reference genome is not possible and we shall assume no prior knowledge about the genome of that organism or its length, or composition. Thus, de novo genome assembly comes into play. It can also be used for a species with a solved reference genome for variant discovery if we need to avoid any bias created by a prior knowledge. De novo genome assembly is a strategy to assemble a novel genome from scratch without the aid of a reference genome sequence. Because of the improvements in cost and quality of DNA sequencing, de novo genome assembly now is widely used specially in metagenomics for bacterial and viral genome assembly from environmental and clinical samples.

The de novo genome assembly aims to join reads into a contiguous sequence called a contig. Multiple contigs are joined together to form a scaffold and multiple scaffolds can also be linked to form a chromosome. The genome assembly is made of the consensus sequences. Both single-end and paired-end or mate-pair reads can be used in the de novo assembly, but paired reads are preferred because they provide high-quality alignments across DNA regions containing repetitive sequences and produce long contigs by filling gaps in the consensus sequence. Assembling the entire genome is usually challenging because of the presence of numerous stretched tandem repeats in the genome. These repeats create gaps in the assembly. Gap problem can be overcome by deep sequencing, which is sequencing a genome multiple times to provide sufficient coverage and sequence depth, which increase the chance for read overlaps. The sequencing coverage is defined as the average number of reads that align to or cover known reference bases of the genome and it is estimated as follows [1]:

$$\text{coverage} = \frac{\text{read length} \times \text{number of reads}}{\text{haploid genome length (bp)}} \quad (3.1)$$

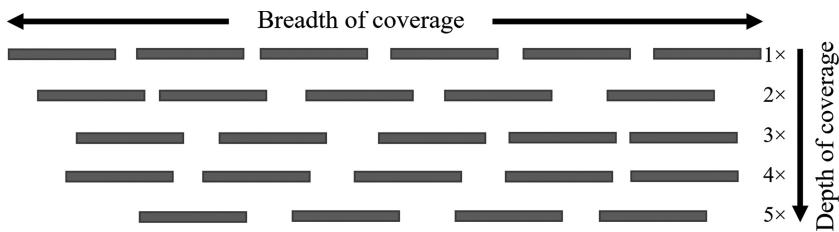


FIGURE 3.1 Sequencing coverage and depth.

The coverage describes how often, on average, a reference sequence is covered by bases from the reads. The sequencing depth is defined as the total number of reads that align to or cover a specific locus (Figure 3.1).

The de novo assembly strategy has been greatly enhanced by the single-molecule real-time (SMRT) sequencing (Pacific Biosciences (PacBio)) and nanopore sequencing (Oxford Nanopore Technologies (ONT)), which produce long reads. In general, the de novo genome assembly depends on the read length, sequencing depth, and the assembling algorithm. Based on the sequencing technology used, reads can be classified into short reads, which are produced by the most sequencing technologies including Illumina, and long reads produced by PacBio and ONT. In general, either sequence alignment or graphs approach is used for the de novo assembly. The algorithms used by the de novo genome assemblers can be classified into (i) greedy approach, (ii) overlap-layout-consensus with Hamiltonian path, or (iii) de Bruijn graph with Eulerian path. The last two approaches share the idea that a genome assembly can be represented as a path in graphs.

3.1.1 Greedy Algorithm

The greedy algorithm was the first one used for de novo genome assembly. Like multiple sequence alignment, it depends on the similarity between reads to create a pileup of aligned sequences, which are collapsed to create contigs from the consensus sequences. The greedy algorithm uses pairwise alignment to compare all reads to identify the most similar reads with sufficient overlaps and then it joins them together. The reads with the significant overlaps are merged. The algorithm continues by conducting pairwise alignments on the merged reads to identify the sequences with significant overlaps and merges them. This process will continue repeatedly until there is no more merging. When there is low sequencing depth, gaps may be present between the assembled contigs; deep sequencing and the use of paired-end reads can reduce gaps and allow longer contigs. Greedy assemblers include TIGR assembler [2] and PHRAP [3]. For instance, assume that we have the reads CATTG, ATTG, GTCAT, TTCGC, and GGCAT. Figure 3.2 shows how the greedy algorithm works to assemble the consensus sequence GTCATTGCGAT from those reads.

3.1.2 Overlap-Consensus Graphs

The overlap graph method was first used for assembling genomes from sequences produced by Sanger sequencing method and then it was extended by some assemblers for NGS reads.

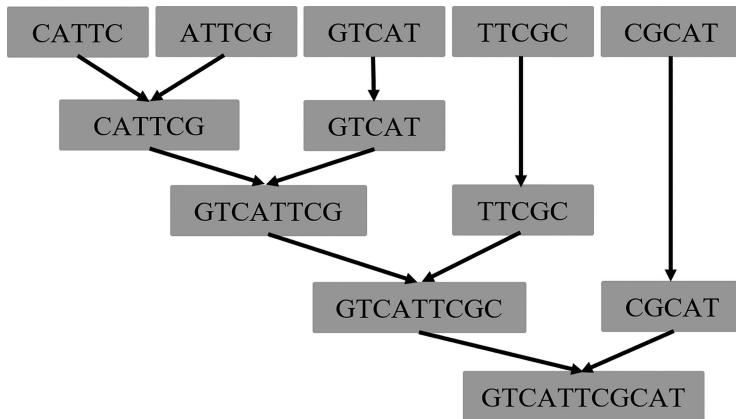


FIGURE 3.2 Greedy assembly.

The overlap-consensus algorithm performs pairwise alignment and then it represents the reads and the overlaps between the reads with graphs, where contiguous reads are the vertexes (nodes) and their overlaps (common prefix or suffix) are the edges. Each node (r_i) corresponds to a read and any two reads are connected by an edge $e(r_1, r_2)$ where the suffix of the first read matches the prefix of the second read. The algorithm then attempts to find the Hamiltonian path of the graph that includes the vertexes (reads) exactly once. Contigs are then created from the consensus sequences of the overlapped suffixes and prefixes. The downside of this method is that it requires a large sequencing coverage and the complexity of the Hamiltonian path increases with the increase of reads. An example for an assembler using this approach is Celera Assembler [4]. To show you how the approach of the overlap-consensus graphs works, assume that we have the reads:

```

CAGAACCTAGC
ATAGCAGAACG
GCTAAGCAGTG
AGTGTACGAC
TAGCAACTATG
AACGTAGCAGA
  
```

Figure 3.3 and 3.4 show the overlap graphs in which the reads are the nodes and the overlapped prefixes and suffixes are the edges. The Hamiltonian path includes each node exactly once to form the consensus sequence.

3.1.3 De Bruijn Graphs

De Bruijn graph [5, 6] approach for de novo assembly was first introduced in 2001 for assembling short reads produced by the NGS technologies and then it was refined in 2008 [7]. In de Bruijn graphs, each read is broken into overlapping substrings of length k called overlapping k -mers. The k -mers then are represented by graphs in which each k -mer is a vertex (node). Any two nodes of any two k -mers that share a common prefix and suffix of length

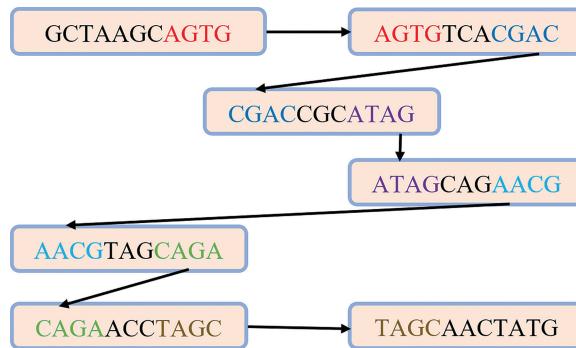


FIGURE 3.3 Overlap graphs and Hamiltonian path.

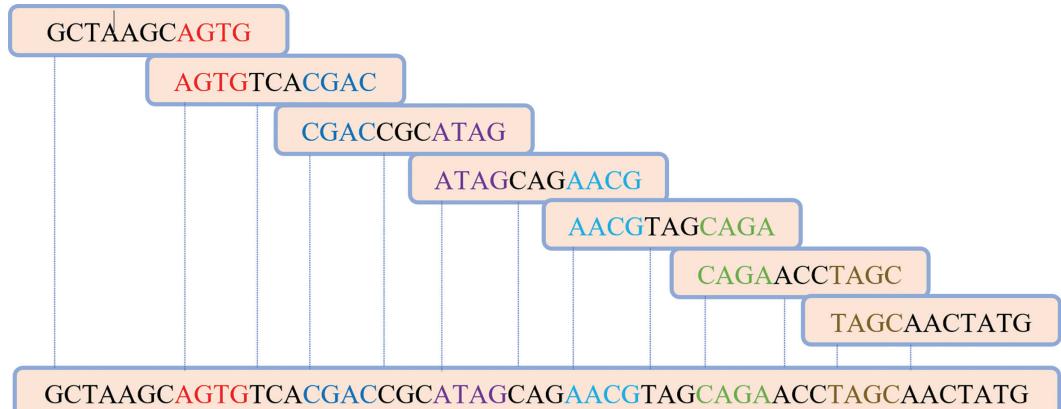


FIGURE 3.4 The overlaps and consensus sequence.

$k-1$ are connected by an edge. The contiguous reads are merged by finding Eulerian path, which includes every edge exactly once. Most modern assemblers like ABySS and SPAdes use de Bruijn graphs method because it is less complex than the overlap graphs. The quality of the de novo assembly is greatly affected by the value of the parameter k (number of bases). For instance, assume that we have the read TCTACTAAGCTAACATGCAATGCA. When we implement de Bruijn algorithm in any programming languages, we can obtain the graphs showing the nodes (k -mers) and the edges as shown in Figure 3.5.

For both overlap and de Bruijn graphs, once we have constructed the graphs, the next step is to find the relevant paths that are likely to represent the assembly. There should be a single path for the assembly if there is no error or sequence repeats, which is rare in most cases.

3-mers and frequencies <pre>'TCT': 1, 'CTA': 3, 'TAC': 1, 'ACT': 1, 'TAA': 2, 'AAG': 1, 'AGC': 1, 'GCT': 1, 'AAC': 1, 'ACA': 1, 'CAT': 1, 'ATG': 2, 'TGC': 2, 'GCA': 2, 'CAA': 1, 'AAT': 1</pre>	De Bruijn Graphs (edges and nodes) <pre>{'TCTA': 1, 'CTAC': 1, 'TACT': 1, 'ACTA': 1, 'CTAA': 2, 'TAAG': 1, 'AAGC': 1, 'AGCT': 1, 'GCTA': 1, 'TAAC': 1, 'AACG': 1, 'ACAT': 1, 'CATG': 1, 'ATGC': 2, 'TGCA': 2, 'GCAA': 1, 'CAAT': 1, 'AATG': 1, 'GCAT': 1, 'CATC': 1, 'ATCT': 1} {('AAC', 'ACA'), ('AAG', 'AGC'), ('AAT', 'ATG'), ('ACA', 'CAT'), ('ACT', 'CTA'), ('AGC', 'GCT'), ('ATC', 'TCT'), ('ATG', 'TGC'), ('CAA', 'AAT'), ('CAT', 'ATC'), ('CAT', 'ATG'), ('CTA', 'TAA'), ('CTA', 'TAC'), ('GCA', 'CAA'), ('GCA', 'CAT'), ('GCT', 'CTA'), ('TAA', 'AAC'), ('TAA', 'AAG'), ('TAC', 'ACT'), ('TCT', 'CTA'), ('TGC', 'GCA')}</pre>
	The plot of de Bruijn graphs of the read

FIGURE 3.5 De Bruijn graphs.

3.2 EXAMPLES OF DE NOVO ASSEMBLERS

The above is a brief definition for the three kinds of the algorithms for the de novo genome assembly. For the algorithms themselves, you may need to refer to an algorithm book. In the following, we will discuss some example assemblers to show you how the de novo genome assembly is performed. There are many papers that reported the comparative performance of these assemblers and others. Readers can refer to those papers to see the differences found by researchers.

3.2.1 ABySS

ABySS (Assembly By Short Sequences) [8] is a parallel de novo genome assembler developed to assemble very large data of short reads produced by NGS technologies. It performs assembly in two stages. First, it generates all possible k-mers from the reads, removes potential errors, and builds contigs using de Bruijn graphs. Second, it uses mate-pair information to extend contigs benefiting from contig overlaps and merges the unambiguously connected graph nodes. ABySS can be used in two modes: bloom filter mode, which uses hashing, and MPI mode, which uses message passing interface (MPI) to parallelize the de novo assembly. It is recommended to use the bloom filter mode over the legacy MPI because it reduces the memory usage to 10 folds. ABySS can be installed following the instructions available at “<https://github.com/bcgsc/abyss>”. On Ubuntu, we can install it using “`sudo apt-get install abyss`”. Once ABySS has been installed, the “`abyss-pe`” command can be

used for the de novo assembly. For the purpose of the practice, we will use paired-end short reads produced by Illumina MiSeq for whole genome sequencing of *Escherichia coli* (str. K-12). The files (forward and reverse FASTQ files) are available at the NCBI SRA database. First, using the Linux terminal, create a directory for the exercise and change to it.

```
mkdir denovo; cd denovo
```

Inside that directory, create a subdirectory for the FASTQ files.

```
mkdir fastq; cd fastq
```

Then, you can download the raw data files into “fastq” directory using SRA toolkits as follows:

```
fasterq-dump --threads 4 --verbose ERR1007381
```

To save some storage space, you can compress the two FASTQ files with GZIP utility as follows:

```
gzip ERR1007381_1.fastq
gzip ERR1007381_2.fastq
```

The compression will reduce the storage of the two files from 11 G to 3 G.

We will use “abyss-pe” command to perform the de novo genome assembly. Change to the main exercise directory just a single step out of “fastq” directory by using “cd ..”. Then, run the following command to construct contigs:

```
abyss-pe \
  name=ecoli \
  j=4 \
  k=25 \
  c=360 \
  e=2 \
  s=200 \
  v=-v \
  in='fastq/ERR1007381_1.fastq.gz fastq/ERR1007381_2.fastq.gz' \
  contigs \
  2>&1 | tee abyss.log
```

The following will construct scaffolds from the contigs:

```
abyss-pe \
  name=ecoli \
  j=4 \
  k=25 \
  c=360 \
```

```
e=2 \
s=200 \
v=-v \
in='fastq/ERR1007381_1.fastq.gz fastq/ERR1007381_2.fastq.gz' \
scaffolds stats
```

The “name=ecoli” specifies the prefix string as “ecoli”, “j=4” specifies the number of processors to be used, “k=25” specifies the length of the k-mer (substring), “c=255” specifies the minimum mean k-mer coverage of a contig (a high-confidence contig), “e=2” specifies the minimum erosion k-mer coverage, “s=200” specifies the minimum contig size (bp) required for building scaffolds, “v=-v” enables verbose, and “in=” specifies the FASTQ file names as an inputs.

When the execution of the above commands on the Linux terminal is complete, the assembly statistics will be displayed. Assembling a genome with “abyss-pe” command is performed in three stages: assembling contigs without paired-end information, aligning the paired-end reads to an initial assembly, and finally merging contigs joined by paired-end information. Multiple files will be generated for those three stages. The descriptions of these files are as follows:

- A file with “*.dot” extension is a graph file in Graphviz format in which graphs (nodes and edges) are defined in DOT language script. We can visualize this file by drawing the graphs with the Graphviz program, which can be installed on Linux using “sudo apt install graphviz”. We can then draw the graphs and save it in the PNG format using:

```
dot -Tpng ecoli-5.dot -o ecoli-5.png
```

- A file with “*.hist” extension is a histogram file made of two tab-separated columns: the fragment size and count. It shows the distribution of the fragment sizes.
- A file with “*.path” extension is an ABySS graph path file, which describes how sequences should be joined to form new sequences.
- A file with “*.sam.gz” extension is a SAM file that is used by ABySS to describe alignments of reads to assemble sequences at different stages of the assembly.
- A file with “*.fa” extension is a FASTA file format for contig and scaffold sequences. The definition line of the FASTA file consists of three parts: <SEQ_ID><SEQ_LEN><KMERS>, where SEQ_ID is a unique identifier for the sequence assigned by ABySS, SEQ_LEN is the length of the sequence in bases, and KMERS is the number of KMERS that mapped to the sequence in the assembly.
- A file with “*.fai” extension is an indexing file for the index of the corresponding FASTA sequences. It includes five tab-separated columns (name of the sequence, length of the sequence, offset of the first base in the file, number of bases in each line, and number of bytes in each line).

- A file with “*stats.csv” or “*stats.tab” extension is the statistics of contig/scaffold contiguity in CSV format. The assembly statistics generated by ABySS are shown in Figure 3.6 and described in Table 3.1.

The contig/scaffold N50 metric is the most widely used metric for describing the quality of a genome assembly. A contig/scaffold N50 is calculated by first ordering every contig/scaffold by length from the longest to the shortest. Next, the lengths of contigs are summed starting from the longest contig until the sum equals one-half of the total length of all contigs in the assembly. The contig/scaffold N50 of an assembly is the length (bp) of the shortest contig/scaffold from the sequences that form 50% of the assembly. To compare between assemblies, the longer the N50 and the smaller the L50, the better the assembly.

In Figure 3.6, the scaffolds file (ecoli-scaffolds.fa) contains 836 sequences, of which 107 sequences are more than 500 bp. The shortest sequence has 584 bp and the longest is 267,586 bp. The N50 is the sequence of length 112,320 bp and L50 (the number of scaffolds that accounts for more than 50% of the genome assembly) is 15.

Figure 3.7 shows a diagram explaining the major metrics of the genome assembly (N25=55, N50=70, N75=75, L25=4, L50=6, and L75=7) and how they can be computed. In the figure, there are eight contigs ranked from the smallest to the largest. The total number of bases is 445 Mb (100%) and the half number is 222.5 Mb (50%).

You can display both contigs and scaffolds file on a Linux terminal using the “less” Linux command as:

	n:500	L50	min	N75	N50	N25	E-size	max	sum	name
2111	914	183	504	4320	7428	12327	9097	30395	4480393	ecoli-unitigs.fa
946	198	31	584	26800	43397	77722	53532	138139	4544461	ecoli-contigs.fa
836	107	15	584	55181	112320	163957	114113	267586	4544805	ecoli-scaffolds.fa

FIGURE 3.6 Assembly statistics.

TABLE 3.1 Assembly Statistics

Column	Description
N	The total number of sequences in the FASTA file
n:500	The number of sequences whose lengths are not less than 500 bp
L50	The number of scaffolds that account for more than 50% of the assembly
LG50	The number of scaffolds that account for more than 50% of the genome assembly
NG50	The sequence length of the shortest contig at 50% of the total genome length
Min	The size of the smallest sequence
N75	The sequence length of the shortest contig at 75% of the total assembly length
N50	The sequence length of the shortest contig at 50% of the total assembly length
N25	The sequence length of the shortest contig at 25% of the total genome length
E-size	The sum of the square of the sequence sizes divided by the assembly size
Max	The size of the largest sequence
Sum	The sum of the sequence sizes
Name	The file name of the assembly

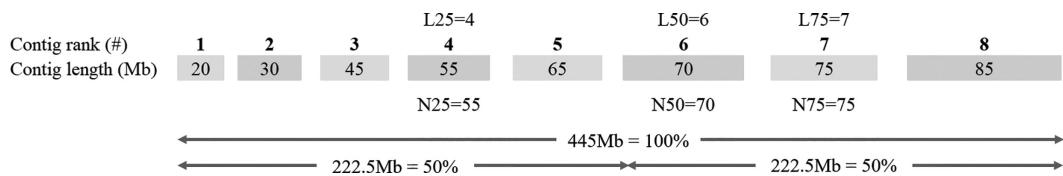


FIGURE 3.7 Genome assembly metrics.

```
less ecoli-contigs.fa  
less ecoli-scaffolds.fa
```

Use “awk” command to print the length of the longest scaffold in the scaffold file.

```
awk '{print length}' ecoli-contigs.fa | sort -n | tail -n1
```

3.2.2 SPAdes

SPAdes [9] is a de novo genome assembler developed primarily for assembling small genomes of bacteria. Later, modules were added for assembling small genomes of other organisms including fungi and viruses. It is not recommended for assembling large mammalian genomes. The current SPAdes version works with both Illumina and Ion Torrent reads, and it can be used for genome hybrid assembly for PacBio, Oxford Nanopore, and Sanger reads. This assembler can process several paired-end and mate-paired files in the same time. The program also provides separate modules for metagenomic data, plasmid assembly from the whole genome sequencing data, plasmid from metagenomic data, transcriptome assembly from RNA-Seq data, biosynthetic gene cluster assembly with paired-end reads, viral genome assembly from RNA viral data, SARS-CoV-2 assembly, and TruSeq barcode assembly. The assembling process of SPAdes includes four stages. First, de Bruijn graphs are built from overlapping k-mers generated from the reads. Second, the k-mers are adjusted to obtain accurate distance estimates between k-mers using both distance histograms and paths in the assembly graphs. The program then constructs paired de Bruijn graphs, which is a generalization of the de Bruijn graph that incorporates mate-pair information into the graph structure [10]. Finally, contigs are constructed from the graphs.

SPAdes program is made up of modules in Python. The installation instructions are available at "<https://cab.spbu.ru/files/release3.15.4/manual.html>". To install the program on Linux, use the following steps:

Using the Linux terminal, first download and decompress the source program in a local directory.

```
 wget https://cab.spbu.ru/files/release3.15.4/SPAdes-3.15.4-Linux.tar.gz  
 tar -xzf SPAdes-3.15.4-Linux.tar.gz
```

Notice that the program name or path may change in the future.

After decompression, the program files will be in “SPAdes-3.15.4-Linux/bin” directory. You will need to add the program path to “.bashrc” file to be able to use the SPAdes program files from any directory.

```
export PATH="/home/your_path/SPAdes-3.15.4-Linux/bin:$PATH"
```

Replace “your_path” with the correct path to the directory on your computer. After adding the line, restart the terminal or use “source ~/bashrc” for the change to take effect. You can test the installation by running the following command on the Linux terminal command line:

```
spades.py --test
```

The above command will assemble toy read data that comes with the program. The output will be saved in “spades_test”, which are the typical SPAdes output files.

Remember that we have downloaded two FASTQ files for *E. coli* and we used them with ABySS above. The two files were saved in “denovo/fastq”; run the following command on the Linux command line while you are in the working directory “denovo”:

```
python spades.py \
--pe1-1 fastq/ERR1007381_1.fastq.gz \
--pe1-2 fastq/ERR1007381_2.fastq.gz \
--isolate \
-o spades_ecoli_ass
```

The above code assembles contigs and scaffolds from the paired-end reads in the input FASTQ files and saves the output files in a new directory “spades_ecoli_ass”. The output files include intermediate file used for the construction of contigs and scaffolds, log files, contigs’ file, and scaffolds’ file in FASTA format. The latter represents the assembly.

SPAdes can perform hybrid de novo assembly using any of PacBio continuous long reads (CLR) or Oxford Nanopore reads as input with Illumina or Ion Torrent reads. You will need to use “--pacbio” option for the PacBio FASTQ file and “--nanopore” option for the Oxford Nanopore FASTQ file. In the following example, first, we download four PacBio SMRT FASTQ files “SRR801646”, “SRR801649”, “SRR801652”, and “SRR801638” for *E. coli* K12. Second, we will use these four PacBio files with the Illumina short read files to assemble *E. coli* genome with SPAdes program.

```
python spades.py \
--pe1-1 fastq/ERR1007381_1.fastq.gz \
--pe1-2 fastq/ERR1007381_2.fastq.gz \
--pacbio pacbio/SRR801646.fastq.gz \
--pacbio pacbio/SRR801649.fastq.gz \
--pacbio pacbio/SRR801652.fastq.gz \
--pacbio pacbio/SRR801638.fastq.gz \
```

```
--careful \
--isolate \
-o hyb_spades_ecoli_ass
```

We used “--gzip” with “fastq-dump” command to compress the FASTQ files.

SPADEs program has different modes for different applications. With “spades.py” command, we can use “--meta” option for metagenomic mode, “--bio” option for biosyntheticSPADEs mode, “--corona” option for coronaSPADEs mode, “--rna” option for transcriptomic mode (RNA-Seq reads), “--plasmid” option for plasmid detection mode, “--metaviral” option for virus detection mode, “--metaplasmid” option for metagenomic plasmid detection mode, and “--rnnaviral” option for virus assembly mode (from RNA-Seq reads).

In the following example, we will download Illumine paired-end FASTQ files “ERR8314890” for the whole genome sequencing of SARS-CoV-2 virus surveillance from the NCBI SRA database. Then, we will use coronaSPADEs module to assemble SARS-CoV-2 genome. For this exercise, you can create a directory “sarscov2” and change into it as:

```
mkdir sarscov2; cd sarscov2
```

Then, you can download the FASTQ files from the NCBI SRA database using SRA toolkits program “fasterq-dump”.

```
fasterq-dump --verbose ERR8314890
```

Then, you can run SPADES program to assemble the SAR-CoV-2 genome using “--corona” option.

```
python spades.py \
--pe1-1 ERR8314890_1.fastq \
--pe1-2 ERR8314890_2.fastq \
--corona \
-o sarscov2_genome
```

The output files including FASTA files of contigs and scaffolds will be saved in the specified output directory “sarscov2_genome”.

The other SPADES modes work the same.

3.3 GENOME ASSEMBLY QUALITY ASSESSMENT

After assembling a genome using any of the de novo genome assemblers, the next step is to assess the quality of assembly to have an idea about how the assembly is good. Genome assessment metrics provide important information on how the assembly is reliable or not. There are two approaches for the quality assessment of an assembly. The first one is a statistical approach that depends on statistical metrics for measuring the quality of a genome

assembly. An example of programs which use the statistical approach is QUAST. The second approach depends on evolutionary relatedness to similar organisms to estimate the number of genes in the genomes and gene completeness. An example of the programs that use this approach is BUSCO.

3.3.1 Statistical Assessment for Genome Assembly

The statistical assessment of a genome assembly can be performed in two ways: reference-guided approach and non-reference approach. One reason for using the *de novo* genome assembly is the unavailability of a reference genome for the species studied. However, we know that the *de novo* assembly can also be used in some applications to assemble a genome of an organism that has a reference genome. When an organism has no reference genome, the *de novo* genome assembly can be assessed without a reference genome. Some assemblers like ABySS can generate some useful statistics for assessing the assembled genome. Table 3.1 includes the statistical metrics for a genome assembly. Not all assemblers have modules generating that assessment metrics. We can use a program like QUAST (QUality ASsessment Tool) [11] for this purpose. The current version of QUAST can be downloaded, decompressed, and installed as follows:

```
wget https://downloads.sourceforge.net/project/quast/quast-5.0.2.tar.gz
tar -xzf quast-5.0.2.tar.gz
cd quast-5.0.2
sudo ./setup.py install_full
```

This will install QUAST and set the path so that you will be able to run it from any directory. Notice that the above file path or name may change in a future version. QUAST can be used to assess genome assemblies (contigs or scaffolds) generated by *de novo* assemblers. It performs reference-guided assessment, which requires a reference genome sequence for the species studied, or non-reference assessment, in which no reference sequence is used. This is usually the case when the organism is unknown. In the following, we will assess the *de novo* *E. coli* genomes assembled with ABySS and SPAdes above without using a reference sequence. You can copy the FASTA scaffold files created by these two programs into a separate directory (e.g., “qc”) with new names using Linux command line as follows:

```
mkdir qc
cp abyss_ecoli_ass/ecoli-scaffolds.fa qc/abyss_ecoli_ass.fasta
cp spades_ecoli_ass/scaffolds.fasta qc/spades_ecoli_ass.fasta
cp hyb_spades_ecoli_ass/scaffolds.fasta qc/spades_hyb_ecoli_ass.fasta
```

With the above commands, we created a directory “qc” and we copied the scaffolds’ files created by ABySS and SPAdes into it.

The next step, change into the new directory “cd qc” and run the following QUAST command to perform quality assessment for the three genome assemblies.

```
quast.py \
-o quast_Ecoli_ass \
-t 4 \
abyss_ecoli_ass.fasta \
spades_ecoli_ass.fasta \
spades_hyb_ecoli_ass.fasta
```

If the following error is displayed:

“/home/.../jsontemplate.py” line 583, in <module> ‘html’: cgi.escape,

You may need to open that file “jsontemplate.py” using a text editor of your choice. In line 50 of that file change “import cgi” to “import html” and in line 583 change ““html”: cgi.escape” to ““html”: html.escape” and rerun the above command.

The above QUAST command saves the output files in “quast_Ecoli_ass” directory. The assessment report is generated in different formats including pdf, html, text, and TSV. You can open any of these reports with the right application. We can open “report.html” on Firefox browser by running the following on the Linux terminal:

```
cd quast_Ecoli_ass
firefox report.html
```

This will display a statistics table, which includes assembly assessment statistics (Figure 3.8), and three interactive plots: cumulative length, Nx, and GC content (two are shown in

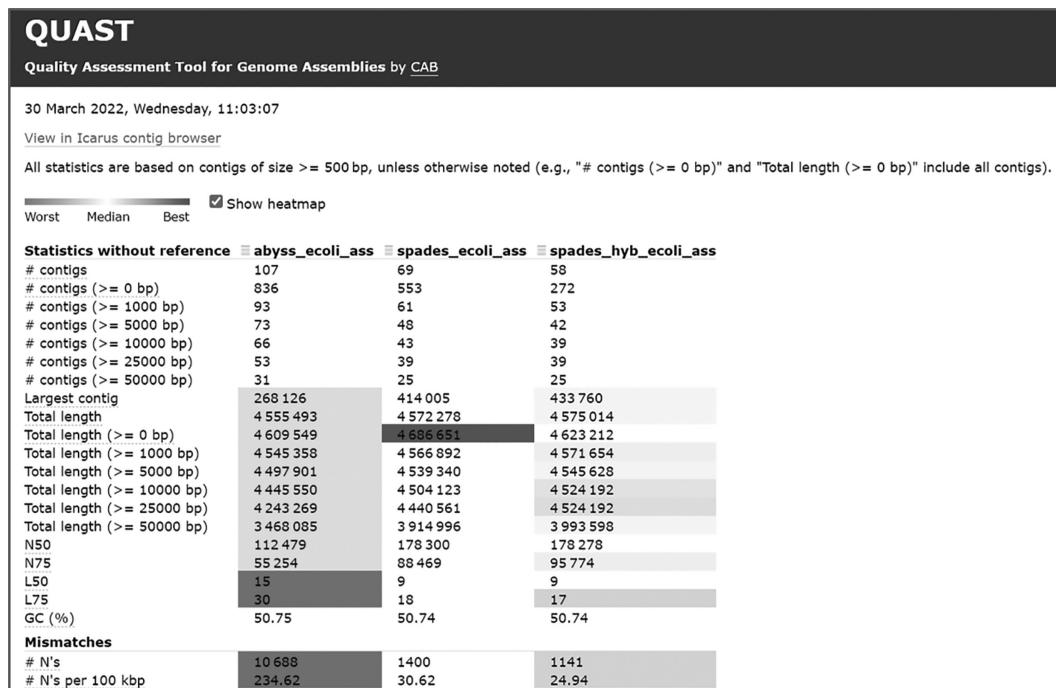


FIGURE 3.8 QUAST assembly assessment report.

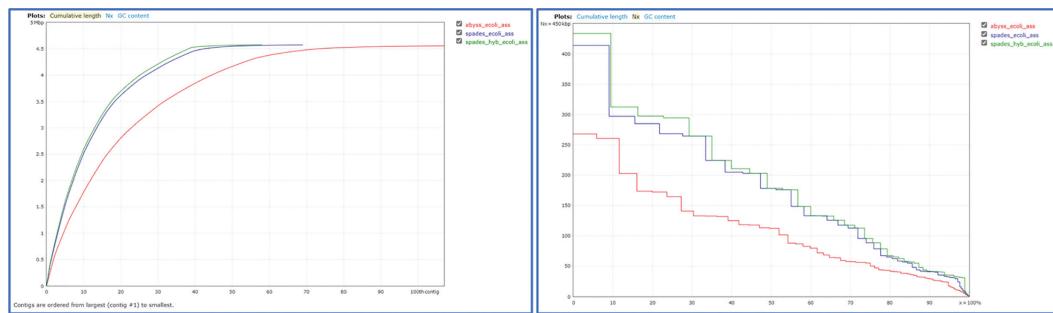


FIGURE 3.9 QUAST assembly assessment plots.

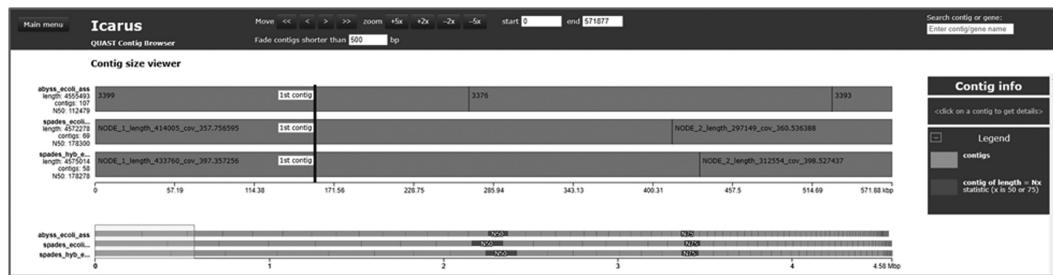


FIGURE 3.10 Icarus contig browser.

Figure 3.9). Use the mouse to display these three plots and also move the mouse to obtain immediate information about the scaffold.

The assembly metrics shown in Figure 3.6 are the ones used for assembly assessment. The important metrics are the total length of the assembly, N50, N75, L50, and L75. Figure 3.8 shows the metrics for three assemblies in three columns. The two assemblies generated by SPAdes (non-hybrid and hybrid) are better than the assembly generated by ABySS. The largest N50 value, the lowest L50 value, and longest assembly are indicative measures of better assembly. The total lengths of the three assemblies are 4,609,549b (4.609549Mb), 4,686,651b (4.686651Mb), and 4,623,212b (4.623212Mb), respectively, which are close to the length of the reference genome of *E. coli* str. K-12, which is 4.641650Mb. Notice also that SPAdes assemblies have the largest N50 and N75 and the lowest L50 and L75.

The QUAST report also includes Icarus contig viewer, which is a genome viewer based on QUAST for the assessment and analysis of genomic draft assemblies. Click “View in Icarus contig browser” on the top right to display the Icarus QUAST contig browser.

In Figure 3.10, the Icarus visualizer shows contig size viewer, on which contigs are ordered from the longest to the shortest, highlights N50, N75 (NG50, NG75) and long contigs larger than a user-specified threshold. To learn more about Icarus viewer and QUAST output reports, refer to the QUAST manual at “<http://cab.cc.spbu.ru/quast/manual.html#sec3.4>”.

If you need to assess the assembly using a reference as a guide, you can download the FASTA file of a reference genome (Genome) with its annotation file (GFF) from a database

(e.g., NCBI genome) for the organism, and use “-r” and “-g” options as follows (you may need to decompress GFF file) (Figures 3.11 and 3.12):

```
quast.py \
-o ref_quast_Ecoli_ass \
-t 4 \
-r ecolref/GCF_000005845.2_ASM584v2_genomic.fna.gz \
-g ecolref/GCF_000005845.2_ASM584v2_genomic.gff \
abyss_ecoli_ass.fasta \
spades_ecoli_ass.fasta \
spades_hyb_ecoli_ass.fasta
```

3.3.2 Evolutionary Assessment for De Novo Genome Assembly

Rather than contig or scaffold length distributions such as N50 and L50, the evolutionary assessment for a genome assembly is based on the completeness of a genome on the evolutionary informed expectation of genes inferred from closely related orthologous groups of sequences. They assess the completeness of a genome assembly in terms of gene content. BUSCO (Benchmarking Universal Single-Copy Orthologs) [12, 13] is an evolutionary-based quality assessment program that uses information of known genes from a database

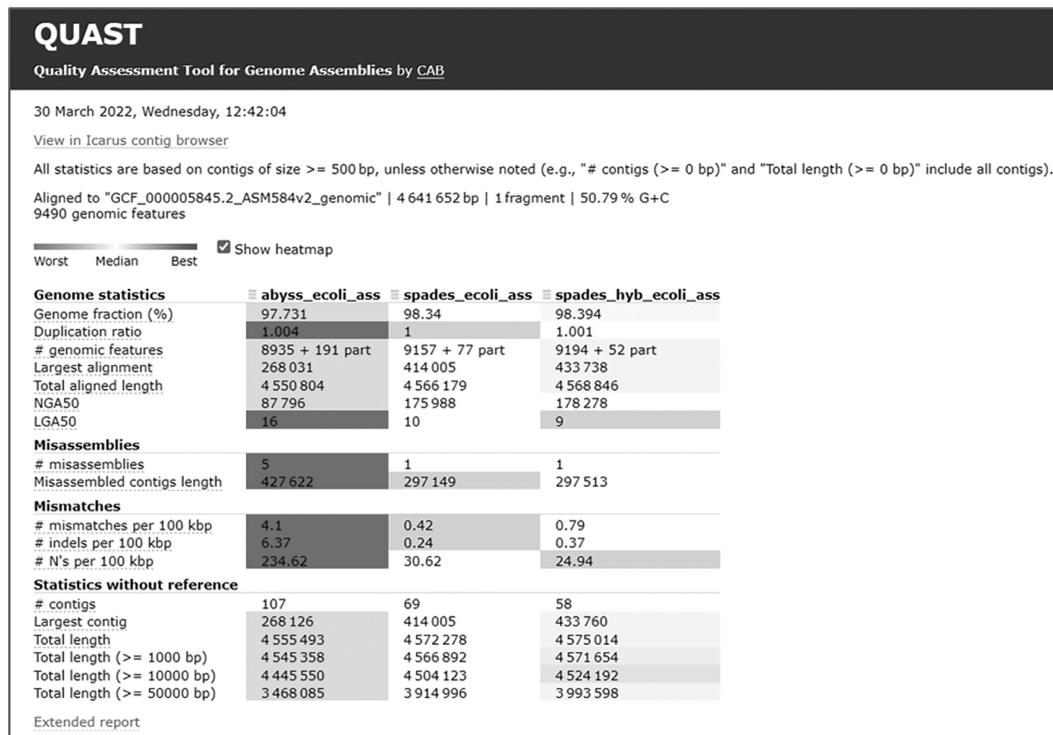


FIGURE 3.11 QUAST assembly assessment report (reference-guided assessment).

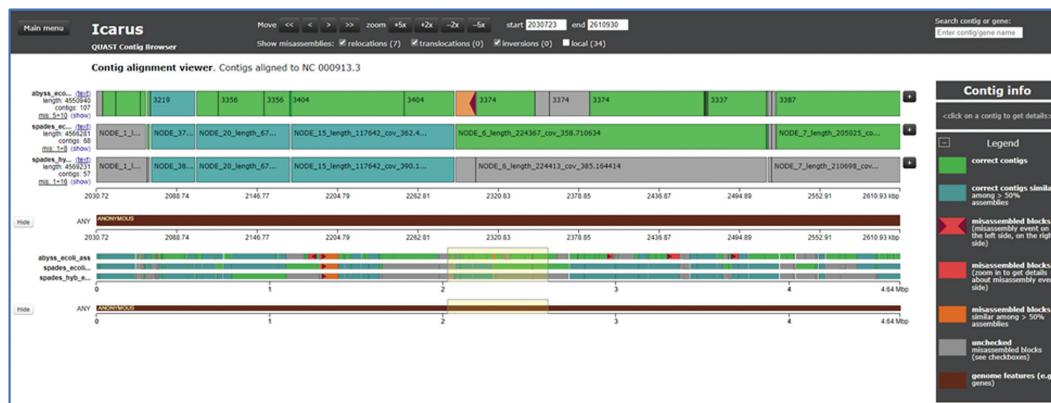


FIGURE 3.12 Icarus contig browser displaying de novo assemblies aligned to a reference genome.

of orthologs (OrthoDB) to measure genome assembly and annotation completeness. The quality of a genome assembly is described by some metrics including *C* for complete, *D* for duplicate, *F* for fragmented, *M* for missing, and *n* for the number of genes used for the assessment. The genes recovered from the de novo assembly are reported as complete (*C*) when their lengths are within two standard deviations of the mean length of genes on the ortholog database. Multiple copies of complete genes are reported as duplicate (*D*), which is an indication of inaccuracy in the assembly of haplotypes. Incomplete or partially recovered genes are reported as fragmented (*F*) and the unrecovered genes are reported missing (*M*). The number of gene used (*n*) reflects the confidence of the assessment results.

BUSCO uses a number of third-party software packages that must be installed for the program to run properly. The BUSCO dependencies include Python 3.x, BioPython, pandas, tBLASTN 2.2+, Augustus 3.2, Prodigal, MetaEuk, HMMER3.1+, SEPP, and R+ggplot2 for the plotting companion script. Some of these packages are needed in some cases. For the complete installation instructions, visit the BUSCO website at “https://busco.ezlab.org/busco_userguide.html”. You can run the following commands on the Linux command line to install some BUSCO third-party dependencies and BUSCO software on Ubuntu:

```
sudo apt update && sudo apt upgrade
pip install biopython
pip install pandas
sudo apt-get install ncbi-blast+
sudo apt install augustus augustus-data augustus-doc
sudo apt install prodigal
sudo apt install hmmer
```

BUSCO software can be cloned and installed by running the following commands:

```
git clone https://gitlab.com/ezlab/busco.git
cd busco/
python3 setup.py install -user
```

If the installation is successful, you can run the following to display the help:

```
busco -help
```

BUSCO databases include ortholog databases for several clades of organisms. Before using BUSCO, you may need to identify the database to use for the assessment. The database list can be displayed using the following command:

```
busco --list-datasets
```

Now, we can use BUSCO to assess the three *E. coli* assemblies (one generated with ABySS and two generated by SPAdes). We can save the output of each assessment in a separate directory.

```
busco \
-i abyss_ecoli_ass.fasta \
-o abyss_ecoli_ass.out \
-l bacteria \
-m genome
```

```
busco \
-i spades_ecoli_ass.fasta \
-o spades_ecoli_ass.out \
-l bacteria \
-m genome
```

```
busco \
-i spades_hyb_ecoli_ass.fasta \
-o spades_hyb_ecoli_ass.out \
-l bacteria \
-m genome
```

The BUSCO assessment output for each assembly will be saved in a separate directory: “abyss_ecoli_ass.out”, “spades_ecoli_ass.out”, and “spades_hyb_ecoli_ass.out”. Each of these directories includes an assessment report as a text file and JSON file, in addition to subdirectories for the predicted genes and used ortholog database.

Comparing between the three assemblies based on BUSCO assessment metrics (Figures 3.13–3.15), the two assemblies generated by SPAdes are better than the one generated by ABySS. A total number of 4085 genomes and 124 genes were used to extract informed expected information. The *E. coli* assembled by SPAdes shows 100% completeness (C:100%), no duplicate (D:0.0%), no fragments (F:0.0%), no missing gene (M:0.0%) out of the 124 genes, whereas the BUSCO assessment report for the assembly generated by SPAdes shows C:98.4% [S:98.4%, D:0.0%], F:1.6%, M:0.0%, n:124, which indicates 98.4% of completeness (122 genes are recovered), 1.6% of fragment (2 partially recovered genes).

Combining both statistical and evolutionary assessment for the de novo assembly will provide a good idea about the quality of the de novo assembled genome.

```

BUSCO version is: 5.3.1
# The lineage dataset is: bacteria_odb10 (Creation date: 2020-03-06, number of genomes: 4085, number of BUSCOs: 124)
# Summarized benchmarking in BUSCO notation for file /home/hamiddafa/denovo/abyss_ecoli_ass.fasta
# BUSCO was run in mode: genome
# Gene predictor used: prodigal

***** Results: *****

C:98.4%[S:98.4%,D:0.0%],F:1.6%,M:0.0%,n:124
122    Complete BUSCOs (C)
122    Complete and single-copy BUSCOs (S)
0      Complete and duplicated BUSCOs (D)
2      Fragmented BUSCOs (F)
0      Missing BUSCOs (M)
124    Total BUSCO groups searched

Dependencies and versions:
hmmer: 3.3
prodigal: 2.6.3

```

FIGURE 3.13 BUSCO assessment for the *E. coli* assembly generated by ABySS (Illumina reads).

```

BUSCO version is: 5.3.1
# The lineage dataset is: bacteria_odb10 (Creation date: 2020-03-06, number of genomes: 4085, number of BUSCOs: 124)
# Summarized benchmarking in BUSCO notation for file /home/hamiddafa/denovo/spades_ecoli_ass.fasta
# BUSCO was run in mode: genome
# Gene predictor used: prodigal

***** Results: *****

C:100.0%[S:100.0%,D:0.0%],F:0.0%,M:0.0%,n:124
124    Complete BUSCOs (C)
124    Complete and single-copy BUSCOs (S)
0      Complete and duplicated BUSCOs (D)
0      Fragmented BUSCOs (F)
0      Missing BUSCOs (M)
124    Total BUSCO groups searched

Dependencies and versions:
hmmer: 3.3
prodigal: 2.6.3

```

FIGURE 3.14 BUSCO assessment for the *E. coli* assembly generated by SPAdes (illumine reads).

```

BUSCO version is: 5.3.1
# The lineage dataset is: bacteria_odb10 (Creation date: 2020-03-06, number of genomes: 4085, number of BUSCOs: 124)
# Summarized benchmarking in BUSCO notation for file /home/hamiddafa/denovo/spades_hyb_ecoli_ass.fasta
# BUSCO was run in mode: genome
# Gene predictor used: prodigal

***** Results: *****

C:100.0%[S:100.0%,D:0.0%],F:0.0%,M:0.0%,n:124
124    Complete BUSCOs (C)
124    Complete and single-copy BUSCOs (S)
0      Complete and duplicated BUSCOs (D)
0      Fragmented BUSCOs (F)
0      Missing BUSCOs (M)
124    Total BUSCO groups searched

Dependencies and versions:
hmmer: 3.3
prodigal: 2.6.3

```

FIGURE 3.15 BUSCO assessment for the *E. coli* assembly generated by SPAdes (Illumine reads).

3.4 SUMMARY

Genome assembly is the process of putting nucleotide reads generated by sequencers into the correct order as in the genome of the organism. Despite the development of sequencing technologies and assembling algorithms, the assembled genomes will remain approximate

for the actual genomes. This is because it is hard to avoid errors in sequencing and also genomes of many organisms including repetitive sequences. But the assembly accuracy can be increased by deep sequencing, paired-end sequencing, and the use of long reads produced by PacBio and Oxford Nanopore Technology. The de novo genome assembly has been widely used for different kinds of organisms, specially in metagenomics for the assembly of bacterial, fungal, and viral genomes.

We can use either greedy alignment approach or graph-based approaches for the de novo genome assembly. The greedy method works as multiple sequence alignment by performing pairwise alignment and merging reads to build up contigs. The graph theory approach can be either overlap-consensus graphs or de Bruijn graphs. In the overlap graphs, reads are represented as nodes and the overlapping regions of the reads as edges. Contigs are built by finding the Hamiltonian path which includes each node once. On the other hand, de Bruijn graphs form k-mers from the reads and the k-mers then are represented as nodes. Contigs are formed by including edges using Eulerian path. De Bruijn graphs are more efficient than overlap graphs. The assemblers that use Bruijn graphs include ABySS for small and large genomes and SPAdes for bacterial, fungal, and viral small genomes. SPAdes program has several modules such as metagenomic module, viral assembly module, and SARS-CoV2. SPAdes can also be used to assemble a genome from hybrid reads such as Illumina reads and PacBio reads or Oxford Nanopore reads.

After assembling, a genome can be assessed using both statistical method and evolutionary method. The statistical method generates the number, lengths, and distributions of contigs. The assembly with few but long contigs is an indicator of the good quality. Metrics used to describe statistical quality are N25, N50, N75, L25, L50, and L75. We can compare the performance of assemblers using these metrics. The evolutionary assessment of an assembly relies on the genomes of the evolutionarily related species to identify the number of genes in the assembled genome. The completeness of the genome assembly depends on the complete identified genes. For statistical quality assessment, we can use QUAST, and for evolutionary assessment, we can use BUSCO.

REFERENCES

1. Lander ES, Waterman MS: Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics* 1988, 2(3):231–239.
2. Pop M, Kosack D: Using the TIGR assembler in shotgun sequencing projects. *Methods Mol Biol* 2004, 255:279–294.
3. de la Bastide M, McCombie WR: Assembling genomic DNA sequences with PHRAP. *Curr Protoc Bioinform* 2007, Chapter 11:Unit11. 14.
4. Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, Flanigan MJ, Kravitz SA, Mobarry CM, Reinert KH, Remington KA et al: A whole-genome assembly of *Drosophila*. *Science* 2000, 287(5461):2196–2204.
5. Pevzner PA, Tang H: Fragment assembly with double-barreled data. *Bioinformatics* 2001, 17(suppl_1):S225–S233.
6. Pevzner PA, Tang H, Waterman MS: An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci U S A* 2001, 98(17):9748–9753.
7. Chaisson MJ, Pevzner PA: Short read fragment assembly of bacterial genomes. *Genome Res* 2008, 18(2):324–330.

8. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I: ABySS: a parallel assembler for short read sequence data. *Genome Res* 2009, 19(6):1117–1123.
9. Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, Kulikov AS, Lesin VM, Nikolenko SI, Pham S, Prjibelski AD et al: SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol* 2012, 19(5):455–477.
10. Medvedev P, Pham S, Chaisson M, Tesler G, Pevzner P: Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *J Comput Biol* 2011, 18(11):1625–1634.
11. Gurevich A, Saveliev V, Vyahhi N, Tesler G: QUAST: quality assessment tool for genome assemblies. *Bioinformatics* 2013, 29(8):1072–1075.
12. Manni M, Berkeley MR, Seppey M, Simão FA, Zdobnov EM: BUSCO update: novel and streamlined workflows along with broader and deeper phylogenetic coverage for scoring of eukaryotic, prokaryotic, and viral genomes. *Mol Biol Evol* 2021, 38(10): 4647–4654.
13. Simão FA, Waterhouse RM, Ioannidis P, Kriventseva EV, Zdobnov EM: BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics* 2015, 31(19):3210–3212.

Variant Discovery

4.1 INTRODUCTION TO GENETIC VARIATIONS

Classical genetics began with the works of Gregor Mendel in the 19th century as a field of biology for studying hereditary in species based on morphology and visible results of reproductive acts. Very recently, genetics has been greatly instrumented by bioinformatics that emerged as a logical consequence of the modern progress in molecular biology, information technology, and the urgent need for handling genome-scale data. Bioinformatics has become a key in each step of genetic analysis and it provides standardized framework for variant discovery and description, which are the essence of the modern genetics.

The general workflow of the research on genetic variation focuses on the analysis and identification of genetic variants associated with specific phenotypes or populations. Mutations are the key source of genetic variation within a species. A mutation is any change in the nucleotide sequences of the genome of a living organism. The basic mutations include base substitution, insertion, and deletion. However, translocation (exchange of segments between chromosome) and copy number variation (CNV), which is multiple copies of DNA segment, are also source of genetic variations. These mutations can occur in parts of the genome. Base substitution is the most frequent one, and it has been thought that it was a random process and it could occur anywhere in a genome with equal probability, but now such thoughts began to change after genomic studies in several organisms showed that mutations are less frequent in important regions of the genome [1]. Therefore, most genetic studies focus on the mutations that affect genes.

The single base substitution is called point mutation and it can be either transition (when a purine is substituted with another purine or a pyrimidine is substituted with another pyrimidine) or transversion (when a purine is substituted with a pyrimidine or a pyrimidine replaces a purine). The base substitution may have a silent consequence if it occurs in the third position of a codon and the result is a synonymous codon that codes for the same amino acid and does not alter the protein sequence. The substitution will have missense consequence if it results in a nonsynonymous codon that codes for a different amino acid and alters the protein polypeptide sequence. The consequence of a substitution,

in this case, can either lead to no change in the structure and function of the protein (conservative mutation) or it may lead to a deleterious consequence if the change alters the protein structure and function. The base substitution may also have a nonsense consequence if it results in a stop codon that truncates the translated protein leading to an incomplete and nonfunctional protein.

A deletion mutation is the removal of a single pair of nucleotides or more from a gene that may result in a frameshift and a garbled message and nonfunctional product. Deletion may have deleterious consequence or not depending on the part it alters and its impact on the protein sequence. The insertion mutation is the insertion of additional base pairs and it may lead to frameshifts depending on whether or not multiples of three base pairs are inserted. Mutations may include combinations of insertions and deletions leading to a variety of outcomes.

In general, a gene variant is a permanent change in the nucleotide sequence of a gene that can be either germline variants, which occur in eggs and sperms of parents and pass to offspring, or somatic variants, which are present only in specific cells and are generally not hereditary.

In terms of sequence change, variants can be classified into single-nucleotide variant (SNV), insertion–deletion (InDel), or structural variation (SV). The SNV is a base substitution of a single nucleotide for another. It is known as single-nucleotide polymorphism (SNP) if its allelic frequency in a population is more than 1%. InDel refers to insertion and/or deletion of nucleotides into genomic DNA and it includes events less than 1000 nucleotides in length. InDels are implicated as the driving mechanism underlying many diseases. The SV involves change in more than 50 base pairs in a sequence of a gene; the change may include rearrangement of part of the genome, a deletion, duplication, insertion, inversion, translocation, or a combination of these. A CNV is a duplication or deletion that changes the number of copies of a particular DNA segment within the genome. SVs have been implicated in a number of health conditions.

In this chapter, we will learn about the major steps in the process of variant identification and analysis, including variant representation, variant calling workflow, and variant annotation. The process by which we identify variants from sequence data (reads) is called variant calling, which is the central topic of this chapter.

4.1.1 VCF File Format

Since a variant is a change in a specific location in a genome, in bioinformatics, this requires a format that can describe the type of a mutation and its position relative to the genome coordinates. Thus, the variant call format (VCF) file [2] was developed to hold the information of a large number of variants and also to hold genotype information of multiple samples in the same position. The VCF file, as shown in Figure 4.1, consists of (i) a metadata section for the meta-information and (ii) a data section for variant data. The VCF file has become the standard file for storing variant information for almost all variant calling programs.

Each line in the metadata section of a VCF file begins with “##”. The metadata lines describe the format and content of a VCF file. This can include information about the

sequencing, variant calling software that created the file, or the reference genome used for determining variants. The first few lines of metadata section describe file format, file date, source program, and the reference used. The metadata section also declares and describes the fields provided at both the site-level (INFO) and sample-level (FORMAT) in the data lines of the data section. For example, the following metadata lines describe the ID, data type, and description of some fields that can be found in the INFO and FORMAT columns in the data section:

```
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
```

The data section begins with a tab-delimited single header line that has eight mandatory fields representing columns for each data line (Table 4.1). The column headers of the data section are as follows:

```
#CHROM POS ID REF ALT QUAL FILTER INFO
```

Only if there is genotype data, then a FORMAT column is declared and followed by unique sample names. All of these column names must be separated by tabs as well. Each line in the data section represents a position in the genome. The data corresponds to the columns specified in the header and must be separated by tabs and ended with a new line. Below are the columns and their expected values. In all cases, MISSING values should be represented by a dot (“.”).

As shown in Figure 4.1, the variants are in chromosome 20 on the reference genome NCBI36 (hg18). The figure shows five positions whose coordinates are 14370, 18330,

TABLE 4.1 VCF File Columns

Column #	Column	Description
1	#CHROM	A chromosome identifier (e.g., 11, chr11, X or chrX)
2	POS	A reference position (sorted numerically in ascending order by chromosome)
3	ID	Variant IDs separated by semicolons (no whitespaces allowed)
4	REF	A reference base (A, C, G, or T). Insertions are represented by a dot
5	ALT	A comma-separated alternate base(s) (A, C, G, or T). Deletions are represented by a dot
6	QUAL	A quality score in a log scale (Phred quality score)
7	FILTER	This indicates which filters failed (semicolon-separated), PASS or MISSING
8	INFO	A site-level information in semicolon-separated name-value format
9	FORMAT	A sample-level field name declarations separated by semicolons

Header	<pre> ##fileformat=VCFv4.2 ##fileDate=20090805 ##source=myImputationProgramV3.1 ##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta ##contig=<D=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x> ##phasing=partial ##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data"> ##INFO=<ID=AF,Number=1,Type=Float,Description="Allele Frequency"> ##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele"> ##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129"> ##INFO=<ID=H2,Number=0,Type=Flag,Description="dbSNP2 membership"> ##FILTER=<ID=q10,Description="Quality below 10"> ##FILTER=<ID=q50,Description="Less than 50% of samples have data"> ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype"> ##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality"> ##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth"> ##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality"> </pre>
	<pre> #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA00001 NA00002 NA00003 20 14370 rs6054257 G A 29 PASS NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ 01:48:1:51,51 11:0:48:8:51,51 1/1:43:5:.., 20 17330 . T A 3 q10 NS=3;DP=11;AF=0.017 GT:GQ:DP:HQ 01:49:3:58,50 01:1:3:5:65,3 0/0:41:3 20 1110696 rs6040355 A G,T 67 PASS NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 11:2:21:6:23,27 21:1:2:0:18,2 2/2:35:4 20 1230237 . T . 47 PASS NS=3;DP=13;AA=T GT:GQ:DP:HQ 01:54:7:56,60 01:0:48:4:51,51 0/0:61:2 20 1234567 microsat1 GTC G,GTCT 50 PASS NS=3;DP=9;AA=G GT:GQ:DP 0/1:35:4 0/2:17:2 1/1:40:3 </pre>

FIGURE 4.1 VCF file showing metadata and data sections.

1110696, 1230237, and 1234567. The first two variants are single point substitutions. The third position has two alternate alleles (G and T) that replaced the ref nucleotide (A). The fourth variant is a deletion of a single nucleotide T since the alt allele is missing (“.”). In the fifth position, there are two alternative alleles, the first is a deletion of two nucleotides (T and C) and the second is an insertion of a single nucleotide T.

The QUAL column holds the quality level of the data at each position. The FILTER column designates what filters can be applied; the keywords in this column can be used to filter the variants as we will discuss later. The second row (position 17330) does not pass the threshold for the quality of more than 10 Phred quality score.

The INFO column includes position-level information for that data row and can be thought as aggregate data that includes all of the sample-level information specified.

The FORMAT column specifies the sample-level fields to expect under each sample. Each row has the same format fields (GT, GQ, DP, and HQ) except for the last row which does not have HQ. Each of these fields is described in the metadata section. GT (Genotype) indicates which alleles separated by / are unphased or | phased, GQ is the Genotype Quality which is a single integer, DP is the Read Depth which is a single integer, and HQ is the Haplotype Quality, and it has two integers separated by a comma.

This VCF file has three samples identified by their names (NA00001, NA00002, and NA00003) in columns 10 through 12.

Genetic variants discovered by researchers are submitted, usually in VCF files, to databases that archive information of the genetic differences with other related information. Researchers submit data to these databases, which collect, organize, and publicly document the evidence supporting links between genetic variants and diseases or conditions. The variants are usually submitted with their assertions, which are informed assessments of the association or lack of association between a disease or condition and a genetic variant based on the current state of knowledge. The variant databases include dbSNP (for human variants of lesser than 50 base pairs), dbVar (for human variants of greater than 50 base pairs), and European Variation Archive (for variants of all species).

Variant submitted to a database is given a unique identifier that can be used in finding that variant in the database and the related information because they are unambiguous,

unique, and stable, unlike descriptive names, which can be used differently by different people. For example, the NCBI dbSNP assigns an ID with “rs” prefix to the accepted human variants with asserted positions mapped to a reference sequence as reference variants (RefSNP) and also it assigns an ID with “ss” prefix for a variant submitted with flanking sequence. Figure 4.1 shows two dbSNP IDs for reference variants in the VCF ID columns. Variants may have identifiers from multiple databases. You will see these different types of identifiers used throughout the literature and in other databases. Different types of identifiers are used for short variants and structural variants.

4.1.2. Variant Calling and Analysis

Variant calling is the process by which we can identify variants on sequence data. The sequence data are usually stored in FASTQ files obtained from whole genome, whole exome sequencing, or targeted gene sequencing. The reads in the FASTQ files are assessed for quality and then preprocessed to ensure that final reads are of high quality. The reads are then aligned to a reference genome and the read alignment information are stored in BAM files. The BAM files are then used as an input for variant calling programs for variant identification and analysis. The identified variants are written in a VCF file. A single VCF file can hold thousands of variants and genotypes of multiple samples. The genetic studies usually focus on the germline variant calling, where the reference genome used for mapping the reads is standard for the species of interest; that will allow us to identify genotypes. The somatic variant calling is used to study diseases like cancer. In somatic variant calling, the reference is a related tissue from the same individual (e.g., healthy tissue in the case of cancer). Here, we expect to see genetic mosaicism between cells or presence of more than one genetic line as a result of genetic mutations.

A variant calling workflow begins with raw sequencing data for multiple samples or individuals and ends with a single VCF file containing only the genomic positions where at least one individual in the population has a variant due to mutations. After variants have been called, they can be analyzed in different ways. For example, we may wish to determine which genes are affected by the variants, what consequences they have on them, and the phenotypes associated with them. Thus, variants that have been called can be annotated with their consequences and can also be associated to certain phenotypes and the results can be interpreted to answer some research questions.

Before digging into the steps of variant calling and analysis, it is better to distinguish between the types of genetic variation studies. There are several types of genetic variation studies but generally they can be classified into (i) Genome-wide association studies (GWASs) [3], (ii) studies on consequences of variants [4], and (iii) Population genetics [5].

The GWASs involve genotyping a sample of individuals at common variants across the genome using a genome-wide survey for variants. Variants associated with a phenotype will be found at a higher frequency. This kind of studies are carried out on individuals to identify variants and their associated phenotypes as variants causing the phenotype will be at higher frequency in the affected individual than in the control. The phenotype–genotype associations must be supported by statistical evidence based on the population studied.

The studies on the consequences of variants focus on understanding the molecular mechanisms and pathways that link a genotype to a phenotype. This kind of studies interpret the consequences of the variants on the protein function. Simple base substitution such as missense, stop gained, and stop lost variants can alter the translated protein sequence, causing functional consequences. Moreover, functional consequences due to structural variants are usually defined by the physiological phenotypes observed. These can be complex descriptions which are described using general phenotypic traits rather than specific biochemical effects caused by the variant. Clinical functional consequences are represented by a simple controlled vocabulary that defines the relative pathogenicity of a variant, such as benign, likely benign, uncertain significance, likely pathogenic, or pathogenic.

The studies of population genetics are the studies of variation within populations of individuals and the forces that shape it. This usually involves studying changes in frequencies of genetic variation in populations over space and time. Some of the major forces that shape variation in natural populations are mutations, selection, migration, and random genetic drift. When a new mutation occurs, it may be beneficial to the organism, deleterious (harmful) to the organism, or it can be neutral (have no effect on the fitness of the organism). Indeed, beneficial and deleterious mutations are subject to natural selection, typically leading to increases and decreases in their allele frequency, respectively. Allele frequencies are also influenced by the random genetic drift. This process explains the fluctuation in allele frequencies from one generation to another.

4.2 VARIANT CALLING PROGRAMS

There are several programs for variant calling using different variant calling algorithms. The most commonly used variant calling programs are categorized into two groups: consensus-based callers like BCFTools mpileup and haplotype-based callers like FreeBayes and GATK HaplotypeCaller. In the following, we will discuss these two types of variant callers with some examples. We will assume that the FASTQ files used in the exercise are preprocessed and clean as explained in Chapter 1.

4.2.1 Consensus-Based Variant Callers

The consensus-based variant callers depend on the pileup of the aligned reads covering a position on the reference sequence to call the variants (SNVs or InDels). The read alignment information is in SAM/BAM file. We can then check the pileup of all bases of the reads covering a reference base position. In most cases, the bases covering that position will be the same as the base of the reference sequence, but in the case of variants, the bases will be different from the reference base. The consensus sequence is created by collapsing bases on all position and choosing the most frequent bases. In some positions, there may be differences between the sequence of the reference genome and consensus sequence. These differences can also be due to errors; however, when there is a sufficient sequencing depth, that will provide sufficient confidence to call the variants. Figure 4.2 shows a diagram for reads aligned to the sequence of a reference genome and a consensus sequence formed by

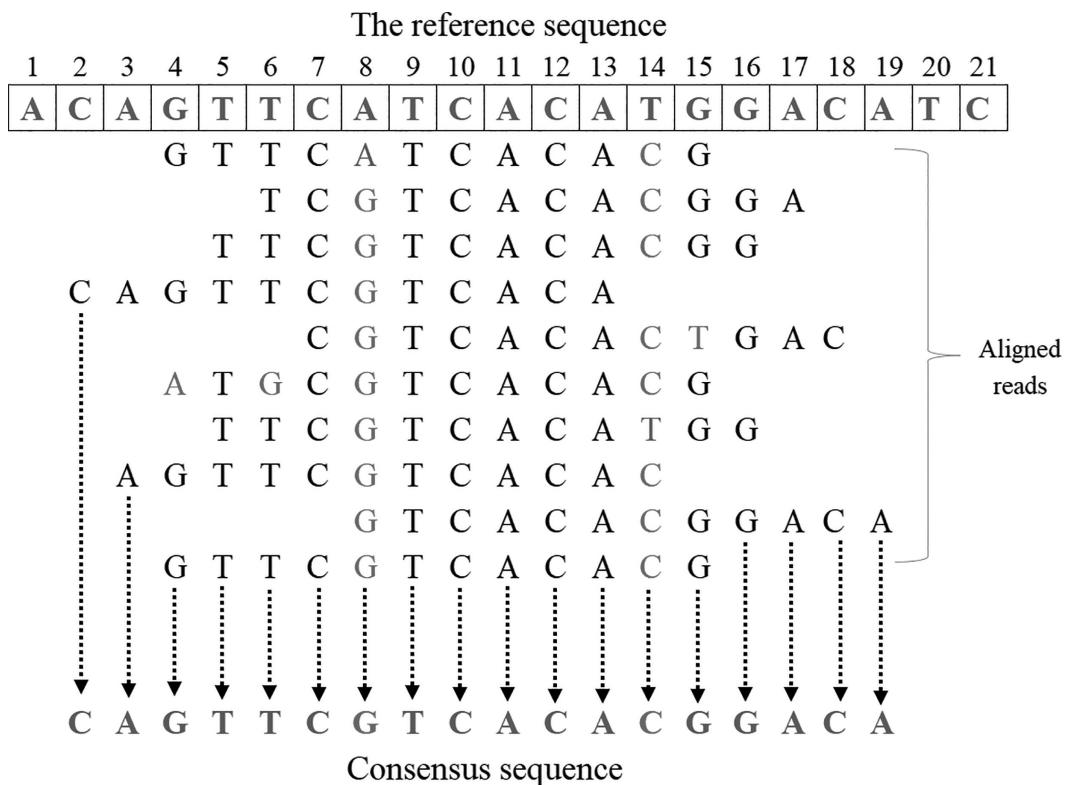


FIGURE 4.2 Read alignment and a consensus sequence.

collapsing bases in each position and identifying the most frequent base. In the figure, you can notice how depth is important in determining the consensus base.

4.2.1.1 BCF Tools Variant Calling Pipeline

In the following, we will use BCFtools as a consensus variant caller to pileup and call variants. BCFtools [6] is a set of tools for the manipulation of variant in VCF files and its binary form BCF. To download the latest BCFtools version, follow the installation instructions available at "<http://www.htslib.org/>". On Linux and Linux like platforms, you can also use the following commands to install it:

```
sudo apt-get update -y  
sudo apt-get install -y bcftools
```

Run the following commands on the Linux terminal to learn about the usage and options of the "bcftools" and "bcftools mpileup":

```
bcftools --help  
bcftools mpileup
```

The general form for the variant calling is as follows:

```
bcftools mpileup -Ou \
-f reference_file \
-b bam_list_file \
| bcftools call -vmO z \
-o sarscov2.vcf.gz
```

The “bcftools mpileup” is the command used for the pileup. The “-f” option specifies the reference file used by the aligner to produce the BAM file, “-b” option specifies the BAM file from which we wish to call variants. The above form allows both pileup and variant calling.

In the following, we will discuss an example of variant calling pipeline using “bcftools”. We will use SARS-CoV-2 raw sequence data from the NCBI SRA database to demonstrate variant calling.

SARS-CoV-2 is the virus that causes COVID-19 which affected millions of people around the world and caused thousands of deaths. The virus mutates in a short period of time, producing new strains. The following are the NCBI SRA run unique identifiers of raw data generated by whole genome sequencing of SARS-CoV-2:

```
SRR16989486
SRR16537313
SRR16537315
SRR16537317
```

In the following, we wish to identify variants (SNVs and InDels) from short reads of those four samples and report that in a VCF file.

The programs used with this example include SRA toolkits, wget, gzip, samtools, bwa, and bcftool on Linux.

The workflow for variant discovery includes: (i) acquiring the raw data (FASTQ files), (ii) quality control if required, (iii) downloading and indexing the reference genome, (iv) using an aligner to map reads in FASTQ files to a reference genome to generate a BAM file, (v) sorting the aligned reads in BAM files, (vi) removal of duplicate reads from the BAM files, and (vii) performing variant calling and generation of a single VCF file for genotyping of all samples. Any new generated BAM files must be indexed before proceeding to the next step.

In the following, we will discuss each of these steps. First, we need to open the Linux terminal, create a directory called “sarscov2” for this project, and make it the current working directory.

```
mkdir sarscov2
cd sarscov2
```

4.2.1.1.1 Acquiring the raw data

The raw data files are stored in the NCBI SRA database with above SRA run IDs. The FASTQ files are in sequence read archive (SRA) format that requires SRA toolkits to be downloaded and extracted. For that purpose, we can use “fasterq-dump” with the following syntax:

```
fasterq-dump --progress --outdir fastq id
```

The “--progress” option is to display the downloading progress, “--outdir fastq” specifies the directory where FASTQ files are downloaded, and “id” is replaced by any of the above SRA run IDs.

The above “fasterq-dump” form is suitable for a single run, but what if we have multiple run IDs as above, or in some cases, we may have tens of IDs to download and running that command for each ID would be tedious. In such case, bash “while loop” would come in handy. First, we need to store the above run IDs in the file “ids.txt”, each run ID in a line, and save the file in the current directory and then run the following bash script, which creates the subdirectory “fastq” and then uses “while loop” to loop over each run ID in the text file and use it as an argument for the “fasterq-dump” command as follows:

```
mkdir fastq
while read id;
do
  fasterq-dump --progress --outdir fastq "$id"
done < ids.txt
```

The above script creates the directory “fastq” and downloads the FASTQ files into it. There are two FASTQ files for each sample since the reads are paired end (forward and reverse

```
(base) hamid@node2:~$ mkdir fastq
while read id;
do
  fasterq-dump --progress --outdir fastq "$id"
done < ids.txt
join :|----- 100%
concat :|----- 100%
spots read    : 1,578,272
reads read    : 3,156,544
reads written : 3,156,544
join :|----- 100%
concat :|----- 100%
spots read    : 608,111
reads read    : 1,216,222
reads written : 1,216,222
join :|----- 100%
concat :|----- 100%
spots read    : 783,381
reads read    : 1,566,762
reads written : 1,566,762
join :|----- 100%
concat :|----- 100%
spots read    : 805,671
reads read    : 1,611,342
reads written : 1,611,342
Cannot process .
(base) hamid@node2:~$
```

FIGURE 4.3 Using fasterq-dump to download FASTQ files from the NCBI SRA database.

FASTQ files). When the FASTQ files have been downloaded successfully as shown in Figure 4.3, we can use “ls fastq” to display the files in the new created directory.

4.2.1.1.2 Downloading and indexing the reference genome sequence

The reads in the FASTQ files must be aligned to a reference genome of the organism studied. Therefore, the latest FASTA file of the reference genome sequence is downloaded from a genome database into a local drive. The NCBI Genome database “<https://www.ncbi.nlm.nih.gov/genome/>” is one of the databases that curates reference genome sequences. We can use the database query box to search for the latest reference genome of “SARS-CoV-2” and copy the link to the FASTA sequence of the reference genome. The following bash script creates the “ref” subdirectory, downloads the compressed FASTA sequence of the latest SARS-CoV-2 reference genome into that subdirectory, and decompresses the FASTA file. Notice that the URL of the reference sequence may change if a new version is available. Therefore, visit the reference genome page for the latest sequence. When you use the following script, make sure that “wget” and the URL are in the same line and that there is no whitespace in the URL. After downloading the reference sequence, use “ls” to make sure the file has been downloaded and decompressed.

```
mkdir ref
cd ref
wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/
GCF_009858895.2_ASM985889v3/GCF_009858895.2_ASM985889v3_genomic.
fna.gz
f=$(ls *.*)
gzip -d ${f}
```

4.2.1.1.3 Indexing the FASTA file of the reference genome

As discussed in Chapter 2, we need to index the FASTA sequence of the reference genome with both “samtools faidx” and the aligner used for mapping. In this example, we will use “bwa” aligner; therefore, we will use “bwa index” for indexing as well. The following bash script uses “samtools” and bwa” to index the reference genome:

```
f=$(ls *.*)
samtools faidx ${f}
bwa index ${f}
cd ..
```

When you display the content of the “ref” subdirectory, you may see the following files if you follow the above steps successfully:

```
GCF_009858895.2_ASM985889v3_genomic.fna
GCF_009858895.2_ASM985889v3_genomic.fna.amb
GCF_009858895.2_ASM985889v3_genomic.fna.ann
GCF_009858895.2_ASM985889v3_genomic.fna.bwt
```

```
GCF_009858895.2_ASM985889v3_genomic.fna.fai
GCF_009858895.2_ASM985889v3_genomic.fna.pac
GCF_009858895.2_ASM985889v3_genomic.fna.sa
```

4.2.1.1.4 Aligning short reads of FASTQ files to the reference genome

Since we have downloaded the FASTQ files and the sequence of the reference genome, it is time to map the reads to the reference genome and to create a SAM file for each sample. Remember that since the reads are paired end, there are two files (forward and reverse) for each run. This time we will use the following syntax of “bwa mem” command:

```
bwa mem -M -t 4 \
-R "@RG\tID:....\tSM:...." \
ReferenceSequence \
file1.fastq file2.fastq > output.sam 2> output.log
```

The “-M” option is to mark shorter split hits as secondary, “-t” option specifies the number of threads to speed up the mapping, and “-R” to add a header line. For variant calling, we may need to add or create a read group (RG) header to hold the sample information. As input, we will provide the FASTA reference sequence and the forward FASTQ file and reverse FASTQ file in the case of paired-end reads. The output can be directed to a SAM file.

The above form is suitable for a single run. However, we may have multiple samples and mapping the reads of a single-sample files at a time may be tedious. Instead, we can use a bash script with a loop to align all runs. The following script creates the subdirectory “sam” and then aligns reads of each sample to the reference genome and outputs a SAM file and a log file. When you run the script, make sure that the main project directory is the working directory.

```
mkdir sam
cd fastq
for i in $(ls *.fastq | rev | cut -c 9- | rev | uniq);
do
bwa mem -M -t 4 \
-R "@RG\tID:${i}\tSM:${i}" \
../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
${i}_1.fastq ${i}_2.fastq > \
../sam/${i}.sam 2> ../sam/${i}.log;
done
cd ..
```

To make sure that the read mapping process has been completed successfully, you can display the content of the “sam” directory with “ls sam”. You can also display the content of any of the SAM files by using “less -S SamFileName”.

Notice that we added a RG header “@GR” with “ID” and “SM” to each SAM file to provide the sample information that will be used later by the variant caller to create a genotype column for each sample.

4.2.1.1.5 Converting SAM into BAM format

The “bwa mem” aligner created a SAM file. We can convert the SAM files into a BAM format by running the following script. Make sure that you are running the script from the main project directory.

```
mkdir bam
cd sam
for i in $(ls *.sam | rev | cut -c 5- | rev);
do
samtools view -uS -o ../bam/${i}.bam ${i}.sam
done
cd ..
```

The above script will create the subdirectory “bam” and convert the SAM files into BAM files and stores them in the “bam” subdirectory.

4.2.1.1.6 Sorting and indexing BAM files

The BAM files must be sorted and indexed before the next step of the analysis. The following bash script creates a new subdirectory “sortedbam”, sorts the bam files, and saves the new BAM files with sorted alignment in the new subdirectory:

```
mkdir sortedbam
cd bam
for i in $(ls *.bam);
do
samtools sort -T ../sortedbam/tmp.sort -o ../sortedbam/${i} ${i}
done
cd ..
cd sortedbam
for i in $(ls *.bam);
do
samtools index ${i}
done
cd ..
```

4.2.1.1.7 Marking duplicates

The identical reads mapped to the reference genome may bias variant calling. The duplicate reads are created by the PCR amplification during sequencing. Usually, using small amount of DNA for library preparation may lead to more duplication. Some sequencing protocols use Unique Molecular Identifiers (UMIs) to distinguish between the artifact and natural duplicates. Marking or removing duplicate aligned reads is a common best practice

in whole genome sequencing. However, several studies have shown that retaining PCR and Illumina clustering duplicates does not cause significant artifacts as long as the library complexity is sufficient. The duplicate alignments can be removed with “samtools rmdup”. The following script creates a new subdirectory “dupRemBam” and then removes duplicate alignments from the BAM files and stores the new BAM file in the new subdirectory:

```
mkdir dupRemBam
cd sortedbam
for i in $(ls *.bam);
do
    samtools rmdup ${i} ../dupRemBam/${i} 2> ../dupRemBam/${i}.log
done;
cd ..
```

4.2.1.1.8 Variant calling with bcftools

It is time to use the bcftools for variant calling. The bcftools is a consensus variant caller as discussed above. It takes a single or multiple BAM files as input and outputs a compressed VCF file. The following script creates the subdirectory “variants” in the main project directory and then it uses “bcftools mpileup” for performing pileup and “bcftools call” to call the variants for the piled-up reads:

```
mkdir variants
cd dupRemBam
ls *.bam | rev | rev > bam_list.txt
bcftools mpileup -Ou \
-f ../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
-b bam_list.txt \
| bcftools call -vmo z \
-o ../variants/sarscov2.vcf.gz
cd ..
```

The VCF file that contains the variants (SNPs and InDels) will be saved in “variants” subdirectory. For further analysis, the VCF file can be indexed using “tabix”, which is a generic indexer for TAB-delimited genome position files.

```
tabix variants/sarscov2.vcf.gz
```

We can view the VCF file with “bcftool view” as follows:

```
bcftools view variants/sarscov2.vcf.gz | less -S
```

All steps from downloading the raw data to variant calling can be included in a single bash file that can be executed on the command-line prompt of the Linux terminal. First, create a new directory, make that directory the current working directory, and then create a file with the run ids “ids.txt” as described above. Then, create a bash file “pipeline_bcftools.

sh" and copy and save the following script on it and then execute the bash file as "bash pipeline_bcftools.sh":

```
#!/bin/bash
#Sars-Cov2 variant calling
#-----
#1- download fastq files from the NCBI SRA database
mkdir fastq
while read f;
do
    fasterq-dump --progress --outdir fastq "$f"
done < ids.txt
#2- download and extract the reference genome
mkdir ref
cd ref
wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/
GCF_009858895.2_ASM985889v3/GCF_009858895.2_ASM985889v3_genomic.
fna.gz
#Extract the compressed the reference FASTA file
f=$(ls *.*)
gzip -d ${f}
#3- Index the reference FASTA file using samtools and bwa
f=$(ls *.*)
samtools faidx ${f}
bwa index ${f}
cd ..
#4- Align the fastq reads (multiple samples) to the reference
genome
mkdir sam
cd fastq
for i in $(ls *.fastq | rev | cut -c 9- | rev | uniq);
do
    bwa mem -M -t 4 \
    -R "@RG\tID:${i}\tSM:${i}" \
    ../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
    ${i}_1.fastq ${i}_2.fastq > \
    ./sam/${i}.sam 2> ./sam/${i}.log;
done
cd ..
#5- convert SAM files into BAM files
mkdir bam
cd sam
for i in $(ls *.sam | rev | cut -c 5- | rev);
do
    samtools view -uS -o ../bam/${i}.bam ${i}.sam
done
cd ..
```

```

#6- Sorting and indexing BAM files
#Sorting BAM files
mkdir sortedbam
cd bam
for i in $(ls *.bam) ;
do
    samtools sort -T ../sortedbam/tmp.sort -o ../sortedbam/${i}
${i}
done
cd ..
#Indexing the sorted BAM files
cd sortedbam
for i in $(ls *.bam) ;
do
    samtools index ${i}
done
cd ..
#7- Marking/removing duplicate alignments
mkdir dupRemBam
cd sortedbam
for i in $(ls *.bam) ;
do
    samtools rmdup ${i} ../dupRemBam/${i} 2> ../dupRemBam/${i}.log
done;
cd ..
#8- Alignment pileup and variant calling using bcftools
mkdir variants
cd dupRemBam
for i in $(ls *.bam) ;
do
    samtools index ${i}
done;
ls *.bam | rev | rev > bam_list.txt
bcftools mpileup -Ou \
-f ../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
-b bam_list.txt \
| bcftools call -vm0 z \
-o ../variants/sarscov2.vcf.gz
cd ..
#9- Index the VCF file
tabix variants/sarscov2.vcf.gz
#to view contents
bcftools view variants/sarscov2.vcf.gz | less -S

```

The resulted VCF file, which contains the genotypes of all samples (Figure 4.4), can be further analyzed as we will discuss at the end of this chapter. However, before analysis, the identified variants must be filtered.

A	B	C	D	E	F	G	H	I	J	K	L	M
#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SRR16537313	SRR16537315	SRR16537317	SRR16989486
29	NC_045512.2	103	.	C	T	18.5455	.	DP=36;SGB=0.5761;GT:PL	0/0;0,18,248	0/0;0,15,216	0/0;0,15,201	0/1;54,0,32
30	NC_045512.2	149	.	G	A	12.2179	.	DP=24;SGB=0.5761;GT:PL	0/0;0,12,212	0/0;0,9,170	0/1;48,0,140	0/0;0,3,60
32	NC_045512.2	152	.	G	T	12.2179	.	DP=24;SGB=0.5761;GT:PL	0/0;0,12,208	0/0;0,9,170	0/1;48,0,140	0/0;0,3,60
33	NC_045512.2	156	.	A	G	12.2179	.	DP=24;SGB=0.5761;GT:PL	0/0;0,12,212	0/0;0,9,170	0/1;48,0,140	0/0;0,3,60
34	NC_045512.2	158	.	A	T	129	.	DP=959;VDB=1.1065;GT:PL	0/1;42,0,179	0/1;39,0,120	0/1;44,0,150	0/1;41,0,27
35	NC_045512.2	210	.	G	T	997	.	DP=963;VDB=0;SGB;GT:PL	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0
36	NC_045512.2	241	.	C	T	998	.	DP=938;VDB=0;SGB;GT:PL	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0
37	NC_045512.2	335	.	C	CT	92	.	INDEL;IDV=5;IMF=0;GT:PL	0/0;0,115,255	0/0;0,30,191	0/0;0,2,182	0/1;127,0,110
38	NC_045512.2	487	.	G	T	8.86477	.	DP=136;SGB=0.576 GT:PL	0/0;0,18,234	0/0;0,18,255	0/1;45,0,183	0/0;0,255,255
39	NC_045512.2	506	.	CATGGTCATGTTATGGT	CATGGT	218	.	INDEL;IDV=94;IMF=0;GT:PL	0/0;0,42,255	0/0;0,54,255	0/0;0,36,255	0/1;255,0,255
40	NC_045512.2	513	.	A	G	3.53801	.	DP=66;SGB=0.5761;GT:PL	0/1;39,0,255	0/0;0,27,255	0/0;0,18,255	0/0;0,33,255
41	NC_045512.2	557	.	G	A	3.8305	.	DP=156;SGB=0.576 GT:PL	0/1;39,0,215	0/0;0,33,255	0/0;0,15,216	0/0;0,255,255
42	NC_045512.2	590	.	C	T	154	.	DP=167;VDB=1.6885;GT:PL	0/1;49,0,187	0/1;71,0,255	0/1;75,0,100	0/0;0,94,255
43	NC_045512.2	740	.	G	A	11.8543	.	DP=79;SGB=0.5761;GT:PL	0/1;48,0,158	0/0;0,30,255	0/0;0,27,255	0/0;0,54,255
44	NC_045512.2	811	.	C	T	5.93584	.	DP=77;SGB=0.5761;GT:PL	0/1;42,0,228	0/0;0,33,255	0/0;0,27,255	0/0;0,63,255
45	NC_045512.2	884	.	C	T	997	.	DP=1019;VDB=0;SGF;GT:PL	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0	1/1;255,255,0
46	NC_045512.2	1059	.	C	T	964	.	DP=1011;VDB=0;SGF;GT:PL	0/1;236,0,220	0/1;255,0,255	0/1;255,0,248	0/1;255,0,255

FIGURE 4.4 VCF file displayed in a spreadsheet.

4.2.1.1.9 Filtering variants

Once variants have been called and identified, it is important to filter out low-quality variants before annotation and interpretation. The traditional variant filtering performs filtering by checking QUAL field and some annotations listed in INFO and FORMAT fields of a VCF file such as MQ (Mapping quality), DP (read depth), FS (FisherStrand), and SOR (StrandOddsRatio) and choosing threshold values to filter out the variants that do not meet the criteria. The base call quality (QUAL) is the most used filter parameter. The traditional VCF filtering is performed with bcftools which uses either “bcftools filter” or “bcftools view” with options “-i” and “-e” to filter-in or filter-out variants. The FILTER field of the VCF file is empty “.” before forcing any filter as shown in Figure 4.4. Once a filter has been applied, a variant in the VCF file either passes or fails. If it passes, PASS will be added to FILTER field and any other variant that does not meet the criterion will be filtered out leaving only the ones with PASS (see Figure 4.5). The following command filters in the variants with Phred quality scores greater than 60:

```
bcftools filter -O z \
-o filtered_sarscov2.vcf.gz \
-i '%QUAL>60' sarscov2.vcf.gz
```

The same results can be obtained with the following command, which filters out variants with quality score less than 60:

```
bcftools view -O z \
-o filtered_sarscov2.vcf.gz \
-e 'QUAL<=60' sarscov2.vcf.gz
```

However, we can also filter variants based on other criteria using the statistics or annotations in INFO or FORMAT fields. For example, you may decide to filter out variants with depth less than 300; thus, you can use the following command:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SRR16537313	SRR16537315	SRR16537317	SRR16989486	
31	NC_045512.2	158	.	A	T	129	PASS	DP=959;VDB=1.1;GT:PL	0/1:42,0,179	0/1:39,0,120	0/1:44,0,150	0/1:41,0,27	
32	NC_045512.2	210	.	G	T	997	PASS	DP=963;VDB=0;SG:GT:PL	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	
34	NC_045512.2	241	.	C	T	998	PASS	DP=938;VDB=0;SG:GT:PL	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	
35	NC_045512.2	335	.	C	CT	92	PASS	INDEL;DV=5;IMF:GT:PL	0/0:115,255	0/0:30,191	0/0:2,182	0/1:127,0,110	
36	NC_045512.2	506	.	CATGGTCATGTTATGGT	CATGGT	218	PASS	INDEL;DV=94;IMF:GT:PL	0/0:42,255	0/0:54,255	0/0:36,255	0/1:255,0,255	
37	NC_045512.2	590	.	C	T	154	PASS	DP=167;VDB=1.6;GT:PL	0/1:48,0,187	0/1:71,0,255	0/1:75,0,100	0/0:94,255	
38	NC_045512.2	884	.	C	T	997	PASS	DP=1019;VDB=0;GT:PL	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	1/1:255,255,0	
39	NC_045512.2	1059	.	C	T	964	PASS	DP=1011;VDB=0;GT:PL	0/1:236,0,220	0/1:255,0,255	0/1:255,0,248	0/1:255,0,255	
40	NC_045512.2	1098	.	A	G	147	PASS	DP=1012;VDB=0;GT:PL	0/0:255,255	0/0:255,255	0/0:255,255	0/1:183,0,255	
41	NC_045512.2	1440	.	G	A	278	PASS	DP=76;VDB=2,40;GT:PL	0/1:94,0,255	0/1:35,0,255	0/1:128,0,175	0/1:64,0,255	
42	NC_045512.2	1666	.	T	C	850	PASS	DP=1012;VDB=0;GT:PL	0/1:215,0,255	0/1:231,0,255	0/1:227,0,255	0/1:215,0,255	
43	NC_045512.2	2043	.	A	T	119	PASS	DP=1000;VDB=1;GT:PL	0/1:40,0,87	0/1:40,0,96	0/1:37,0,156	0/1:41,0,199	
44	NC_045512.2	2073	.	G	C	484	PASS	DP=1010;VDB=0;GT:PL	0/1:94,0,242	0/0:18,223	0/1:95,0,255	0/1:234,0,255	
45	NC_045512.2	3037	.	C	T	965	PASS	DP=998;VDB=4.5;GT:PL	0/1:255,0,187	0/1:255,0,213	0/1:253,0,245	0/1:240,0,255	
46	NC_045512.2	3249	.	A	G	122	PASS	DP=97;VDB=1.60;GT:PL	0/0:0,75,255	0/0:0,75,255	0/0:0,69,230	0/1:158,0,204	

FIGURE 4.5 VCF file containing filtered variants.

```
bcftools filter -O z \
-o filtered2_sarscov2.vcf.gz \
-i 'DP>300' filtered_sarscov2.vcf.gz
```

You can open the filtered VCF file to notice the changes and that the filter command will be added to the VCF header.

Usually, you can implement different filters on the variants in a VCF file to achieve accurate and reliable results.

Another way to filter variants is to use “bcftools isec” with truth variants in a VCF file as input together with your raw VCF file to create intersections, unions, and complements of the VCF files.

```
bcftools isec -c both -p isec truth.vcf.gz input.vcf.gz
```

Refer to bcftools help for more details.

4.2.2 Haplotype-Based Variant Callers

The haplotype-based variant calling programs usually use Bayesian probabilistic model to predict variants on aligned reads based on a haplotype structure of variants rather than only sequence alignment. The haplotype is a set of genetic variants that are inherited together. The haplotype-based variant detection depends on the physical phasing, which is the process of inferring haplotype structure based on genotypic data using the Bayesian approach. The prediction is based on relating the probability of a specific genotype given a set of reads to the likelihood of sequencing errors in the reads and the prior likelihood of specific genotypes. The Bayesian haplotype-based approach allows modeling multiallelic loci that enables direct detection of a longer, multi-base alleles from sequence alignment.

Using reads aligned to a reference sequence (BAM) as input, haplotype-based algorithm first attempts to identify the active regions of variations on the reads aligned to the reference genome. The identification of the active regions is carried out with dynamic sliding windows of certain size along the reference sequence. The number of events (mismatches, InDels, and soft clips) is counted in each window. When the number of events in that

window passes a threshold, then that window will be identified as an active region. A measure like entropy may be used to measure the activity on the region.

The haplotypes are constructed from the reassembled reads following the identification of the active regions. The de Bruijn-like graph is used to reassemble the active region and to identify the possible haplotypes present in the alignments. Once the haplotypes are determined, the original alignment of the reads will be ignored and the candidate haplotypes are realigned to the haplotypes of the reference genome using the Smith-Waterman local alignment. The pairwise alignment is also performed using Pairwise Hidden Markov Model (PairHMM) which generates a likelihood matrix of haplotypes given. These likelihoods are then marginalized to obtain the likelihoods of alleles for each potentially variant site given the read data. The genotype or the most likely pair of alleles is then determined for each position. For a given genotype (G_i) on a subset of overlapped reads (R_i), the variant callers then use Bayesian statistics to evaluate the posterior probability of the hypothetic phenotype (G_i) as follows:

$$P(G_i|R_i) = \frac{P(R_i|G_i) \times P(G_i)}{P(R_i)} \quad (4.1)$$

where the posterior probability $P(G_i|R_i)$ is the probability of the phenotype (G_i) given that subset of reads (R_i), $P(G_i)$ is the prior probability that we expect to observe the genotype based on previous observations, $P(R_i)$ is the probability of the subset of the reads being true (the probability of observing the evidence), and $P(R_i|G_i)$ is the probability of reads given the genotype. The Bayesian variant caller writes the above formula as:

$$P(G_i|R_i) = \frac{P(R_i|G_i) \times P(G_i)}{\sum_k P(R_i|R_k) \times P(G_k)} \quad (4.2)$$

We can ignore the denominator because it is the same for all genotypes. Thus

$$P(G_i|R_i) \propto P(R_i|G_i) \times P(G_i) \quad (4.3)$$

The variant callers use a flat prior probability that can be changed by the users if the probabilities of the genotypes are known based on previous observations. The important probability in the above formula is $P(R_i|G_i)$, which can also be described in terms of the likelihood of the hypothesis of the genotype (G_i) given the reads (R_i):

$$P(R_i|G_i) = L(G_i|R_i) = \prod_j \left(\frac{L(R_j|H_1)}{2} + \frac{L(R_j|H_2)}{2} \right) \quad (4.4)$$

where $L(R_j|H_1)$ and $L(R_j|H_2)$ are the haplotype likelihoods.

The likelihoods of all possible genotypes are calculated based on the alleles that were observed at the site, considering every possible combination of alleles. Then, the most likely

genotype is called and assigned to the sample, and positions with putative variants (substitutions or InDels) are written into the VCF file.

In the following, we will perform variant calling with both FreeBayes and GATK, which are examples of haplotype-based variant callers.

4.2.2.1 FreeBayes Variant Calling Pipeline

FreeBayes [7] is a haplotype-based and Bayesian variant detector that is used to detect small variants such as single and multiple-nucleotide polymorphisms and InDels. On Linux, we can install FreeBayes as follows:

```
sudo apt update
sudo apt install freebayes
```

Use the following command to read more about FreeBayes usage and options:

```
freebayes -help
```

For variant calling, we will use the following form:

```
freebayes \
-f ./ref/GCF_009858895.2_ASM985889v3_genomic.fna \
-C 5 \
-L bam_list.txt \
-v ./variants/sarscov2.vcf
```

The “-f” option specifies the reference file, “-C” specifies the minimum number of observations supporting an alternate allele within a single individual in order to evaluate the position, “-L” will pass the name of the text file that contains the names of the BAM files (each file name in a line), “-v” will pass the VCF file name.

We will use FreeBayes in the above example to identify variants in the SARS-CoV-2 samples. We will follow the same steps we did for “bcftools” above. First, we will create a project directory and store the run IDs in a file “ids.txt” as above. Then, we will save the following script in a file “pipeline_freebayes.sh” and execute it as “bash pipeline_freebayes.sh”:

```
#!/bin/bash
#Sars-Cov2 variant calling
#-----
#1- download fastq files from the NCBI SRA database
mkdir fastq
while read f;
do
    fastq-dump --progress --outdir fastq "$f"
done < ids.txt
```

```
#2- download and extract the reference genome
mkdir ref
cd ref
wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/
GCF_009858895.2_ASM985889v3/GCF_009858895.2_ASM985889v3_genomic.
fna.gz
#Extract the compressed the reference FASTA file
f=$(ls *.*)
gzip -d ${f}
#3- Index the reference FASTA file using samtools and bwa
f=$(ls *.*)
samtools faidx ${f}
bwa index ${f}
cd ..
#4- Align the fastq reads (multiple samples) to the reference
genome
mkdir sam
cd fastq
for i in $(ls *.fastq | rev | cut -c 9- | rev | uniq); do
    bwa mem -M -t 4 \
    -R "@RG\tID:${i}\tSM:${i}" \
    ../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
    ${i}_1.fastq ${i}_2.fastq > \
    ../sam/${i}.sam 2> ../sam/${i}.log;
done
cd ..
#5- convert SAM files into BAM files
mkdir bam
cd sam
for i in $(ls *.sam | rev | cut -c 5- | rev); do
    samtools view -uS -o ../bam/${i}.bam ${i}.sam
done
cd ..
#6- Sorting and indexing BAM files
#Sorting BAM files
mkdir sortedbam
cd bam
for i in $(ls *.bam); do
    samtools sort -T ../sortedbam/tmp.sort -o ../sortedbam/${i}
${i}
done
cd ..
#Indexing the sorted BAM files
cd sortedbam
```

```

cd ..
#7- Marking/removing duplicate alignments
mkdir dupRemBam
cd sortedbam
for i in $(ls *.bam) ;
do
    samtools rmdup ${i} ../dupRemBam/${i} 2> ../dupRemBam/${i}.log
done;
cd ..
#8- Alignment pileup and variant calling using bcftools
mkdir variants
cd dupRemBam
for i in $(ls *.bam) ;
do
    samtools index ${i}
done;
ls *.bam | rev | rev > bam_list.txt
freebayes \
    -f ../ref/GCF_009858895.2_ASM985889v3_genomic.fna \
    -C 5 \
    -L bam_list.txt \
    -v ../variants/sarscov2.vcf
cd ..
#9- Compress and index the VCF file
bgzip variants/sarscov2.vcf
tabix variants/sarscov2.vcf.gz
#to view contents
bcftools view variants/sarscov2.vcf.gz | less -S

```

We may notice that there is a little difference between the VCF file generated by bcftools and the one generated by FreeBayes. The reason is that each program uses a different algorithm for variant calling as discussed above.

4.2.2.2 GATK Variant Calling Pipeline

Genome Analysis Toolkit (GATK) [8] is a collection of command-line tools for variant discovery and analysis of the sequence data acquired from the high-throughput sequencing instruments. For installing the latest release of GATK, follow the installation instructions available at the GATK website “<https://gatk.broadinstitute.org/>”. GATK uses the haplotype approach as FreeBayes. However, it uses advanced workflow for variant calling. The GATK tools can be used individually or together as a complete pipeline for variant discovery. GATK also provides complete workflows called GATK Best Practices for variant calling tailored for specific cases. In the following, we will discuss the most commonly used variant calling pipeline for GATK best practice. The following workflow assumes that the acquired raw data in FASTQ format and that the quality control has been performed on the data. The general GATK workflow consists of the following major steps:

1. Acquiring the raw data in FASTQ files.
2. Downloading the sequence of the reference genome, indexing (with samtools and bwa) the reference sequence, and creating a dictionary (with Picard) for the reference sequence.
3. Mapping the raw reads in FASTQ files to a reference genome using any of the aligner (e.g., BWA) to generate aligned reads in SAM files.
4. Converting SAM files to BAM files using samtools.
5. Sorting and indexing the alignment in the BAM files using samtools.
6. Extracting the chromosome or the interval of interest, otherwise, skipping this step if you are interested in the variants of the whole genome.
7. Removing or marking duplicate alignments in the BAM files using Picard and indexing the resulted BAM with samtools.
8. Adding RG header to the BAM files using Picard and indexing the resulted BAM files with samtools.
9. Applying Base Quality Score Recalibration (BQSR) for detecting systematic errors made by the sequencer.
10. Identification of the genotype at all positions per sample. This step involves “HaplotypeCaller”, which takes a BAM file as input and outputs a GVCF file format for each sample.
11. The next step in the GATK workflow is to consolidate the GVCF files of the multiple samples and the variants are called across all samples in the cohort forming a single VCF file. This step involves GenomicsDBImport and GenotypeGVCFs.
12. Separation of SNPs and InDels in separate VCF files using GATK SelectVariants.

4.2.2.2.1 Acquiring raw data

The raw sequence data usually is in FASTQ format. This time, we will use human raw data from 1000 Genome project to demonstrate variant calling using GATK4 best practice. For simplicity, instead of the whole genome, we will extract only the reads of chromosome 21 of 13 individuals including males and females from Africa, Asia, and Europe. Table 4.2 shows the NCBI SRA data from which the FASTQ files for chromosome 21 will be extracted and sample information.

To keep the files organized, we can create a directory for this project “vcfgatk”, and inside the directory, we will create a text file “ids.txt” that contains only the above ID column without a column header and each ID in a line. Then, while you are inside the project directory, you can save the following bash script in a file “download.sh” and execute it as “bash download.sh”:

TABLE 4.2 The NCBI SRA Run IDs and Individual Information

Run ID	Country	Population	Gender
ERR1019055	China	East Asia	Female
ERR1019056	China	East Asia	Male
ERR1019057	China	East Asia	Female
ERR1019081	Pakistan	South Asia	Male
ERR1025616	Pakistan	South Asia	Male
ERR1019044	Kenya	Africa	Female
ERR1025600	Kenya	Africa	Female
ERR1025621	Nigeria	Africa	Male
ERR1025640	Senegal	Africa	Male
ERR1019034	Russia	Europe	Male
ERR1019045	France	Europe	Male
ERR1019068	Italy	Europe	Male
ERR1025614	Bulgaria	Europe	Male

```
mkdir fastq
cd fastq
while read id;
do
sam-dump \
--verbose \
--fastq \
--aligned-region chr21 \
--output-file ${id}_chr21.fastq \
${id}; \
done < ../ids.txt
cd ..
```

The download may take a long time depending on the speed of the Internet connection and computer memory and processing units. The above script will create the “fastq” directory and save the FASTQ files of chromosome 21 of 13 human individuals.

4.2.2.2.2 The reference genome

The FASTA sequence of the reference genome is required for reads mapping. We can download it from a reliable database such as the NCBI Genome database. However, for GATK pipeline, the sequence of the human genome can be downloaded from GATK resource bundle, which is a collection of standard files prepared to be used with GATK. The resource bundle is hosted on a Google Cloud bucket and can be accessed with a google account using the following address:

<https://console.cloud.google.com/storage/browser/genomics-public-data/resources/broad/hg38/v0/>

With the FASTA sequence, we can also download the sequence dictionary file. Once we have downloaded the FASTA sequence of the reference genome, we can index the sequence with both samtools and bwa. The following script first creates the directory “refgenome”, and then, in that directory, it downloads the sequence of the current human genome and its dictionary file from GATK resource bundle and indexes the FASTA:

```
mkdir refgenome
cd refgenome
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.fasta
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.dict
#or create a fasta dict for reference
f=$(ls *.fasta)
samtools faidx ${f}
bwa index ${f}
cd ..
```

Only human sequences are available in the GATK resource bundle. However, if you download the sequence of a reference genome from a different database, you may need to create a dictionary for that sequence using Picard software as follows:

```
java -jar ~/software/picard.jar \
    CreateSequenceDictionary \
    R=Reference.fasta \
    O=Reference.dict
```

4.2.2.3 Mapping reads to the reference genome

The second step is to align the raw reads to the reference genome that we have downloaded and indexed in the previous step. We can use the aligner of our choice. The most commonly used one is “bwa mem”. The following script creates the directory “sam” and saves the SAM files that contain the read mapping information into that directory. There will be a SAM file for each individual.

```
mkdir sam
cd fastq
ref=$(ls ../*refgenome/*.fasta)
for i in $(ls *.fastq | rev | cut -c 13- | rev);
do
    bwa mem -M -t 4 \
    -R "@RG\tID:${i}\tSM:${i}" \
    ${ref} \
    ${i}_chr21.fastq > \
    ..../sam/${i}_chr21.sam 2> ..../sam/${i}_chr21.log;
done
cd ..
```

4.2.2.2.4 Converting SAM files into BAM files

We must convert SAM files into BAM files to save storage space. Moreover, BAM files can be manipulated faster. The following script creates the directory “bam” and converts the SAM files into BAM files:

```
mkdir bam
cd sam
for i in $(ls *.sam | rev | cut -c 5- | rev);
do
    samtools view -uS -o ../bam/${i}.bam ${i}.sam
done
cd ..
```

4.2.2.2.5 Sorting and indexing alignments in the BAM files

The alignments in BAM files are to be sorted by chromosomes in the reference genome to be used in the downstream analysis. The following bash script uses samtools to sort and index the BAM files and stores them in a new directory called “sortedbam”:

```
mkdir sortedbam
cd bam
for i in $(ls *.bam);
do
    samtools sort -T ./sortedbam/tmp.sort -o ./sortedbam/${i} ${i}
    samtools index ./sortedbam/${i}
done
cd ..
```

4.2.2.2.6 Extracting a chromosome or an interval

Most of the time, we may be interested in the identification of variants on the whole genome. However, sometimes the study may focus on a specific chromosome or an interval of the genome. In case the target is the variants of the whole genome, you can skip this step. You should remember that identifying variants from whole genome requires large memory and storage space. Therefore, for demonstration and the sake of simplicity, we will focus only on chromosome 21 of human genome. In the following, we will create a directory “chr21” and use samtools to extract the alignments of chromosome 21 and store the BAM files and sort them:

```
mkdir chr21
cd sortedbam
for i in $(ls *.bam|rev|cut -c 5-|rev);
do
    samtools view -b ${i}.bam chr21 > ../chr21/${i}.bam
    samtools index ../chr21/${i}.bam
done
cd ..
```

4.2.2.2.7 Mark the duplicate reads

GATK4 best practice recommends removing or marking the duplicate reads mapped to the reference genome to avoid biases in variant calling. In the above examples, we used samtools to mark the duplicate reads. With GATK4 best practice, we can use Picard to mark the duplicate. Picard is a set of command-line tools for manipulating sequencing data such as SAM/BAM and VCF. To install Picard, follow the instructions at “<https://broadinstitute.github.io/picard/>”.

The Picard MarkDuplicates function works by comparing reads in the 5' positions and collecting the duplicates. The function identifies the primary and duplicate reads. The duplicates are marked with the hexadecimal value of 0x0400. For more details, check out Picard manual.

The following script creates a directory called “dedup”, marks the duplicate alignments in BAM files and indexes the resulted BAM files, and stores them in the new directory:

```
mkdir dedup
cd chr21
for i in $(ls *.bam|rev|cut -c 5-|rev) ;
do
    java -Xmx7g -jar ~/software/picard.jar MarkDuplicates \
        -INPUT ${i}.bam \
        -OUTPUT ../dedup/${i}.dedup.bam \
        -METRICS_FILE ../dedup/${i}.dedup_metrics.txt
    samtools index ../dedup/${i}.dedup.bam
done
cd ..
```

4.2.2.2.8 Adding read grouping header to each BAM file

RGs are identified in the BAM file by a number of tags that are defined according to a specific format. We can use the following samtools command to display the RG tags on a BAM file:

```
samtools view -H filename.bam | grep "@RG"
```

GATK4 RGs' tags allow us to differentiate samples and technical features that are associated with artifacts. Such information is used to reduce the effects of artifacts during the duplicate marking and base recalibration steps. GATK requires several RG fields to be present in input files and will fail with errors if this requirement is not met. The RG fields are usually set when the reads are aligned to a reference genome by the aligner. However, if the BAM files do not include the RG fields, we can use the “AddOrReplaceReadGroups” Picard function to add or modify the RG fields as we will do next. AddOrReplaceReadGroups uses the following arguments to add or modify the RG fields on a BAM file:

RGID: This adds group read ID, which must be unique across the samples.

RGPU: This argument is optional for GATK, and it holds the flow cell barcode, lane ID, and sample barcode separated by a period “.”.

RGSM: This holds the sample name that will be used later as a header of sample column in the VCF file.

RGLP: This argument sets the RG field which holds the technology name used for sequencing. The value can be ILLUMINA, SOLID, LS454, HELICOS, or PACBIO.

RGLB: This sets the RG field which holds the DNA preparation library identifier. The “MarkDuplicates” function of GATK uses this field to determine which RGs contain duplicates.

The following script creates the directory “RG” and uses Picard to add the RG to each BAM file. We use the run ID as RG and sample number.

```
mkdir RG
cd dedup
for i in $(ls *.bam|rev|cut -c 5-|rev) ;
do
    java -jar ~/software/picard.jar AddOrReplaceReadGroups \
        I=${i}.bam \
        O=../RG/${i}.RG.bam \
        RGID=${i} \
        RGLB=lib RGPL=ILLUMINA \
        SORT_ORDER=coordinate \
        RGPU=bar1 RGSM=${i}
    samtools index ../RG/${i}.RG.bam
done
cd ..
```

4.2.2.2.9 Building a model for the BQSR

We already know that the raw data may have systematic errors that may affect reporting of the base calling quality score. Such error may lead to overestimate or underestimate the reported quality score. The quality of variant calling basically depends on the quality scores of the base calling that will also affect the read alignment. To minimize the effect of the systematic errors on variant calling, a BQSR is implemented by GATK4 best practice. The BQSR is a machine learning-based method that uses training data to model the empirically observed errors and adjust the quality scores of the aligned reads using that model. The adjusted scores are then used by the variant caller to take decision about a variant calling. The BQSR is achieved in two steps: (i) using a set of known variants as a training dataset for building the recalibration table (with BaseRecalibrator GATK4 function) and (ii) adjusting the base quality scores (with ApplyBQSR GATK4 function). The first step of recalibration process generates a table indicating which sites of the BAM file need adjustment of quality score. The second step of the recalibration process applies recalibration or adjusting the quality scores. The BQSR generates a new BAM file with recalibrated quality scores that variant calling process can rely on. Moreover, the known variants are used to mark the bases at the sites of real variation to avoid being ignored as artifacts. The model training requires high-quality variant datasets (SNPs and InDels) in VCF files downloaded from a reliable source such as NCBI database. Human variant VCF files can also be downloaded from GATK resource bundle as mentioned above.

The following bash script downloads a standard human SNPs and InDels VCF files with their index files in the “refvcf” subdirectory and then it performs the two steps of BQSR. Notice that for BaseRecalibrator tool, the known variant files are provided in “--known-sites” option. The outputs of the base recalibration are stored in “applyBQSR” subdirectory.

```

mkdir refvcf
cd refvcf
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/1000G_phase1.snps.high_confidence.hg38.
vcf.gz
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/1000G_phase1.snps.high_confidence.hg38.
vcf.gz.tbi
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.known_indels.vcf.
gz
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.known_indels.vcf.
gz.tbi
cd ..
##B- Build the BQSR model
#-----
mkdir BQSR
cd RG
ref=$(ls ../refgenome/*.fasta)
for i in $(ls *.bam|rev|cut -c 5-|rev);
do
~/software/gatk-4.2.3.0/gatk --java-options \
-Xmx4g BaseRecalibrator \
-I ${i}.bam \
-R ${ref} \
--known-sites ../refvcf/1000G_phase1.snps.high_confidence.
hg38.vcf.gz \
--known-sites ../refvcf/Homo_sapiens_assembly38.known_
indels.vcf.gz \
-O ../BQSR/${i}.table
done
cd ..
##C- Apply the model to adjust the base quality scores
#-----
mkdir applyBQSR
cd RG
ref=$(ls ../refgenome/*.fasta)
for i in $(ls *.bam|rev|cut -c 5-|rev);
do
~/software/gatk-4.2.3.0/gatk \
--java-options \

```

```

-Xmx4g ApplyBQSR \
-I ${i}.bam \
-R ${ref} \
--bqsr-recal-file ../BQSR/${i}.table \
-O ../applyBQSR/${i}.bqsr.bam
done
cd ..

```

4.2.2.2.10 Variant calling

After BQSR of BAM files, we can perform variant calling on each sample using the “HaplotypeCaller” GATK4 function, which identifies the variation active regions, constructs possible haplotypes using de Bruijn-like graph, and then uses Bayesian theory to call variants as described above. HaplotypeCaller will generate a Genome Variant Call Format (gVCF) file for each sample. The gVCF stores sequencing information for both variant and non-variant sites on a genome sequence. It can hold representation of genotype, annotation, and other information across all sites in the genome in a compact format. Storing sample variant in gVCF format will make consolidation of variants across samples easy.

The following script uses HaplotypeCaller to generate gVCF file for each sample. Notice that since we are targeting only chromosome 21, we will use “-L chr21” option to restrict variant calling to that chromosome. Also notice that chromosome label may be different (e.g., 21, chromosome21); therefore, view the BAM file to check the right chromosome names. The GATK4 GenotypeGVCFs tool is used to generate gVCFs.

```

mkdir gvcf
cd applyBQSR
ref=$(ls ../refgenome/*.fasta)
for i in $(ls *.bam|rev|cut -c 5-|rev) ;
do
~/software/gatk-4.2.3.0/gatk \
--java-options \
-Xmx10g HaplotypeCaller \
-I ${i}.bam \
-R ${ref} \
-L chr21 \
-ERC GVCF \
-O ../gvcf/${i}.g.vcf.gz
done
cd ..

```

4.2.2.2.11 Consolidating variants across samples

The above script used HaplotypeCaller to generate gVCF file for each sample. The next step is to use GenomicsDBImport tool to import single-sample gVCFs into GenomicsDB and to use GenotypeGVCFs tool to consolidate variants across the sample in a single VCF file. For GenomicsDBImport, the input gVCF file is passed through “-V” option. For multiple gVCF

```

ERR1019034    ERR1019034_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019044    ERR1019044_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019045    ERR1019045_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019055    ERR1019055_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019056    ERR1019056_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019057    ERR1019057_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019068    ERR1019068_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1019081    ERR1019081_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1025600    ERR1025600_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1025614    ERR1025614_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1025616    ERR1025616_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1025621    ERR1025621_chr12.dedup.RG.bqsr.g.vcf.gz
ERR1025640    ERR1025640_chr12.dedup.RG.bqsr.g.vcf.gz

```

FIGURE 4.6 Cohort sample map file.

files, we can use “-V” for each one. Instead of using “-V” option several times for multiple gVCF files, the sample information can be saved in a text file called cohort sample map file. The file then can be passed in “--sample-name-map” option. The cohort sample map file is a plain text file that contains two tab-separated columns; the first column is for the sample IDs and the second column is for the names of the gVCF files. Each sample ID is mapped to a sample file name as shown in Figure 4.6.

The cohort sample map file can be created manually by the user. However, we can also use bash script to create it. The following script creates a cohort sample map file for our 13 samples and the file will be as shown in Figure 4.6:

```

cd gvcf
#a- make file name and absolute path
find "$PWD"/*_chr21.dedup.RG.bqsr.g.vcf.gz -type f -printf '%f
%h/%f\n' > ./tmp.txt
#b- remove _1/2.fastq
awk '{ gsub(/_chr21.dedup.RG.bqsr.g.vcf.gz/,",", $1); print } '
./tmp.txt > ./tmp2.txt
rm ./tmp.txt
#remove space
cat ./tmp2.txt | sed -r 's/^\s+//g' > ./tmp3.txt
rm ./tmp2.txt
#replace comma with tab
sed -e 's/[,]/\t/g' ./tmp3.txt > ./cohort.sample_map
rm ./tmp3.txt

```

Once we have created the cohort sample map file, we can run GenomicsDBImport tool to import gVCF sample files and GenotypeGVCFs tool to consolidate the variants of 13 samples in a single VCF file.

```

#create a database
ref=$(ls ./refgenome/*.fasta)
~/software/gatk-4.2.3.0/gatk \
--java-options -Xmx10g \
GenomicsDBImport \

```

```

-R ${ref} \
--genomicsdb-workspace-path ../gvcf21db \
--batch-size 50 \
--sample-name-map ../cohort.sample_map \
-L chr21 \
--tmp-dir ../tmp \
--reader-threads 4

cd ..
mkdir vcf
ref=$(ls refgenome/*.fasta)
~/software/gatk-4.2.3.0/gatk \
--java-options -Xmx10g \
GenotypeGVCFs \
-R ${ref} \
-V gendb://gvcf21db \
-O vcf/allsamples_chr21.vcf

```

4.2.2.12 Variant separation

The output of the previous step is a single VCF file containing both the identified SNPs and InDels. It is good practice to separate each kind of variants in a VCF file because each type of variants may be used for certain analysis. The “SelectVariants” GATK tool is used to select a subset of variants based on a specific criterion. We can use “-select-type” option to select a specific variant type. The valid types are INDEL, SNP, MIXED, MNP, SYMBOLIC, and NO_VARIATION. The following script stores SNPs and InDels in separate VCF files:

```

#SNPS
~/software/gatk-4.2.3.0/gatk \
--java-options \
-Xmx10g SelectVariants \
-V vcf/allsamples_chr21.vcf \
-select-type SNP \
-O vcf/allsamplesSNP_chr21.vcf

#INDEL
~/software/gatk-4.2.3.0/gatk \
--java-options \
-Xmx4g SelectVariants \
-V vcf/allsamples_chr21.vcf \
-select-type INDEL \
-O vcf/allsamplesIndels_chr21.vcf

```

4.2.2.13 Variant filtering

Rather than the traditional variant filtering discussed with the previous variant callers, GATK4 uses an advanced filtering approach called Variant Quality Score Recalibration (VQSR) to filter variants called in the previous step. This approach is similar to the BQSR discussed above as it uses machine learning to model a variant profile in a high-quality training

dataset from validated data resources, such as 1000 Genomes, OMNI, and hapmap, and then it uses the model to filter out the putative artifacts from the called variants. The application of the model results in assigning a log-odds ratio score (VQSLOD) for each variant that measures how likely that variant is real based on the data used in the training. The VQSLOD is added to the INFO field of the variant. The variants are then filtered based on a threshold. SNPs and InDels are recalibrated separately. The variant calibration and filtering are performed in two steps:

(i) Building of the recalibration model:

The recalibration model is built using VariantRecalibrator tool. The input file for this tool is the variants to be recalibrated “-V” and the known training dataset “--resource”. The latter must be downloaded from a reliable source such as GATK resource bundle. The fitted model is used to estimate the relationship between the probability that whether a variant is true or artifact and continuous covariates that include QD (quality depth), MQ (Mapping quality), and FS (FisherStrand). The VQSLOD is estimated based on Gaussian mixture model whether a variant is true versus being false. Each variant in the input VCF file is assigned a VQSLOD in INFO field of the VCF file and the variants are ranked by VQSLOD. A tranche sensitivity threshold can be provided in “-tranche” as a percentage. Several thresholds can be set. The output of this step is a recalibrated VCF file and other files including tranches, which will be used by ApplyVQSR, and plot files.

```
cd refvcf
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.dbsnp138.vcf
wget https://storage.googleapis.com/genomics-public-data/
resources/broad/hg38/v0/Homo_sapiens_assembly38.dbsnp138.vcf.idx
cd ..
mkdir VQSR
cd vcf
~/software/gatk-4.2.3.0/gatk --java-options \
-Xmx10g VariantRecalibrator \
-R ../refgenome/Homo_sapiens_assembly38.fasta \
-V allsamplesSNP_chr21.vcf \
--trust-all-polymorphic \
-tranche 100.0 \
-tranche 99.95 \
-tranche 99.90 \
-tranche 99.85 \
-tranche 99.80 \
-tranche 99.00 \
-tranche 98.00 \
-tranche 97.00 \
-tranche 90.00 \
--max-gaussians 6 \
--resource:1000G,known=false,training=true,truth=true,prior=10.0 \
\
```

```

./refvcf/1000G_phase1.snps.high_confidence.hg38.vcf.gz \
--resource:dbsnp,known=false,training=true,truth=false,prior=2.0
\
./refvcf/Homo_sapiens_assembly38.dbsnp138.vcf \
-an QD -an MQ -an MQRankSum \
-an ReadPosRankSum -an FS -an SOR \
-mode SNP \
-O ./VQSR/tranches.out \
--tranches-file ./VQSR/vqsrplots.R
cd ..

```

(ii) Applying recalibration rules:

Next is applying the model on the variants using ApplyVQSR by tranche sensitivity thresholds to filter variants and adding PASS to FILTER field of the variants that have VQSLOD above the threshold while the variants with VQSLOD below the threshold are labeled with tranche name.

```

#Apply VQSR
mkdir filteredVCF
~/software/gatk-4.2.3.0/gatk \
--java-options -Xmx4G ApplyVQSR \
-V vcf/allsamplesSNP_chr21.vcf \
-O filteredVCF/humanSNP.vcf \
--truth-sensitivity-filter-level 99.7 \
--tranches-file VQSR/vqsrplots.R \
--recal-file VQSR/tranches.out \
-mode SNP \
--create-output-variant-index true

```

We can follow the same steps for filtering InDels.

The downside of the VQSR approach is that it is not applicable for all organisms, because high-quality variant datasets are available only for human and some model organisms. As an alternative to the VQSR, we can use hard filtering with VariantFiltration tool, which enforces a hard filter for one or more annotations such as QD (quality depth), QUAL (Phred quality score), SOR (StrandOddsRatio), FS (FisherStrand), MQ (Mapping quality), and MQRankSum. For these filters, use the thresholds recommended by GATK4. PASS will be added to FILTER field on the VCF file if a variant passes the filtering; otherwise, the name of the filter will be added instead. GATK4 recommends some filters for both SNPs and InDels. The following scripts are for hard filtering SNPs and InDels.

```

#The hard filtering GATK best practice for SNP
~/software/gatk-4.2.3.0/gatk \
--java-options \
-Xmx10G VariantFiltration \
-V vcf/allsamplesSNP_chr21.vcf \
-filter "QD<2.0" \

```

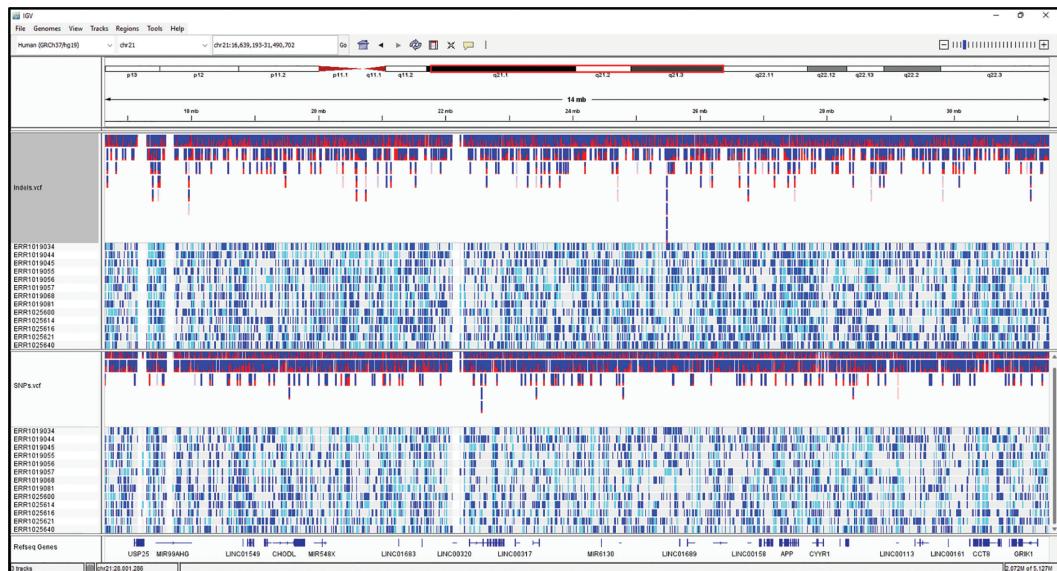


FIGURE 4.7 Visualizing InDels and SNPs with IGV.

```
--filter-name "QD2" \
-filter "QUAL<30.0" \
--filter-name "QUAL30" \
-filter "SOR>3.0" \
--filter-name "SOR3" \
-filter "FS>60.0" \
--filter-name "FS60" \
-filter "MQ<40.0" \
--filter-name "MQ40" \
-filter "MQRankSum<-12.5" \
--filter-name "MQRankSum-12.5" \
-filter "ReadPostRankSum<-8.0" \
--filter-name "ReadPostRankSum-8" \
-O filteredVCF/hardfilteredVCF.vcf

#The hard filtering GATK best practice for INDEL
~/software/gatk-4.2.3.0/gatk \
--java-options \
-Xmx10G VariantFiltration \
-V vcf/allsamplesIndels_chr21.vcf \
-filter "QD<2.0" \
--filter-name "QD2" \
-filter "QUAL<30.0" \
--filter-name "QUAL30" \
-filter "FS>200.0" \
--filter-name "FS200" \
-filter "ReadPostRankSum<-20.0" \
```

```
--filter-name "ReadPostRankSum-20" \
-O filteredVCF/hardfilteredIndels.vcf
```

4.3 VISUALIZING VARIANTS

The variants in a VCF file can be visualized on a variant viewer such as IGV (Integrated Genomics Viewer), which is an open-source program for all platforms. It can be downloaded from “<https://software.broadinstitute.org/software/igv/download>” and installed on a local computer. Figure 4.7 shows the allele fractions and genotypes for each of the InDels and SNPs of the samples. The dark blue color indicates heterozygous genotype, cyan indicates homozygous genotype, and gray indicates the same genotype as the reference. Refer to the documentation of the IGV to read more about this.

4.4 VARIANT ANNOTATION AND PRIORITIZATION

The variant calling using any of the variant callers, such as bcftools, FreeBayes, or GATK, and variant filtering is followed by variant annotation and prioritization. Variant annotation involves adding information and knowledge to high-confidence variants in an effort to enhance assessment of variants that are likely to impact functions. Following the workflow of variant calling, we will obtain high-quality variants in a single VCF file including the genotypes of all samples. Since variant discovery usually involves the whole genome or whole exome of an individual or multiple individuals of a species, thousands of variants may be discovered. However, we are usually interested in the variants that affect the function or have associations with diseases or other important phenotypes. Variants may impact functions in different ways depending on the type of the variants. Variants can be everywhere on the genome sequence, but the most deleterious and damaging are the ones that have effect on the function of a gene. A variant may suppress, inhibit, or activate a gene if it affects the gene regulatory region. This kind of effects are usually seen in cancer cell in which mutations may lead to the hyperactivity of proto-oncogenes, which accelerate cell growth and division or inactivation of tumor suppressor genes. Variants which affect the coding regions of a gene may cause an impact depending on the consequence of the change. A single-nucleotide variant (SNV) that forms a stop codon will cause a truncated protein that does not function normally. On the other hand, SNV in a stop codon may lead to a stop loss that results in a longer protein. A variant in a splicing region may also alter the sequence and function of the protein. More often, SNVs affect the coding regions of a gene producing new amino acids that change the characteristic and function of the translated protein. These SNVs are known as missense SNVs and they are the easiest predictable variants. But an SNV can also be synonymous in the sense that it changes the codon but it does not change the amino acid. Although this kind of variant does not change the protein sequence, it may still have a biological consequence. Insertion or deletion of a single or multiple nucleotides in the coding region may lead to the frameshift, and hence, the protein will be translated incorrectly from that point.

The most deleterious variants are stop-gain, frameshift, and splicing region variants since they lead to loss of function. However, before we decide on a variant effect, we should

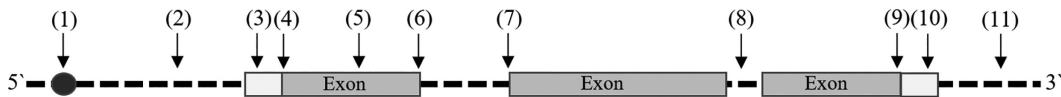


FIGURE 4.8 Variant effect on gene regions.

TABLE 4.3 Gene Regions and Variant Effect

Region	Variant Effect
(1) Regulatory region including transcription factor (TF) binding site	Deleterious variants
(2) Upstream gene region	Intergenic variants/upstream gene variant
(3) 5' UTR region	5' UTR variant
(4) Transcription start site (TSS)	Start retained or start lost variants
(5) Exon region	Exonic variants include missense, nonsense (stop gained), frameshift, inframe insertion or deletion
(6) Splice donor region	Splice-site variant (exon loss, intron inclusion, altered protein-coding sequence)
(7) Splice acceptor region	Splice-site variant (exon loss, intron inclusion, altered protein-coding sequence)
(8) Intron region	Intronic variant
(9) Transcription termination site (TTS)	Stop lost, stop retained, incomplete terminal codon
(10) 3' UTR region	3' UTR variant
(11) Downstream gene region	Downstream gene variant

remember that the consequence may also be beneficial in some cases. For instance, it has been reported that truncating variants in CARD9, IL23R, and RNF186 proteins may protect against Crohn's disease and ulcerative colitis and also truncating variants in ANGPTL4, APOC3, PCSK9, and LPA proteins may protect against coronary heart disease [9, 10]. The impact of variants on a protein is also measured by the number of isoforms produced by the affected gene, the percentage of the protein affected, and moreover, we should put into consideration that a frameshift may be bypassed by splicing or its impact may be avoided by another frameshift. In the attempt to annotate genetic variants, the SNVs effect can be predicted with high accuracy followed by small InDels (1–50 bp) and then medium InDels (50–100 bp). It is also easy to predict the effect of missense SNV. The variant annotation tools have several approaches to predict the effect of missense variant. For instance, they can take into consideration the physicochemical properties of amino acids, whether the variant is in a conserved region or not, or does it affect the three-dimensional structure of the protein. A variant in a region conserved across the species or in a region of a secondary or tertiary structure is more likely to be deleterious. Some tools use homology modeling to simulate the structure of the new protein to predict the effect of variants and other tools use machine learning utilizing multiple features to annotate the variants with the right information and consequences. Figure 4.8 and Table 4.3 show a segment of a eukaryotic genomic gene and possible variant annotations in each region.

In a typical genome-wide variant study, thousands of variants may be discovered. The significance of these variants varies based on the type, location, and possible consequence.

Covering all variants is daunting and hence prioritizing these variants with potential associations to phenotypes of interest is usually the key target of the variant annotation. During the last decade, numerous GWASs were conducted to identify genomic variants associated with many complex diseases and traits. However, most studies were focused on human and some model organisms. Databases for variant mapping and genotype-phenotype association were developed to serve as rich resources for variant annotation. Examples of these databases include NHLBI, which contains health information collected by NHLBI's epidemiological cohorts and clinical trials, dbGaP, which is an NCBI database of Genotypes and Phenotypes association, the Exome Aggregation Consortium (ExAC), which includes sequencing data from a variety of large-scale sequencing projects, Catalogue of Somatic Mutations in Cancer (COSMIC), etc. Numerous similar databases were developed for specific diseases such as cancer, autoimmune diseases, and Alzheimer's disease. Prioritizing genetic variants relevant to the human diseases is the top. Guidelines have been developed for investigating variants and their association with human diseases so that such knowledge can be used for diagnosis in a clinical setting. Indeed, after acquiring high-confidence variants, the next step is to annotate and interpret these variants using either prior knowledge or functional prediction based on the impact of the variant on the translated protein. The studies on genetic variants are usually interested in the variants that are associated with diseases, traits, or have an effect on functions of protein. There are a variety of consequences that can be caused by variants. A variant may be pathogenic or implicated with healthy conditions, or may be a damaging variant that alters the normal function of a gene, or may be deleterious variant that reduces the quality of the affected individuals. Hence, variant annotation must be conducted after filtering variants as discussed above to avoid misinterpretation, false positive, and false negative. Generally, we can define variant annotation as the process of assigning functional or phenotype information to genetic variants such as SNPs, InDels, or copy number variants. Based on this definition, perhaps, the most significant variants are the ones on the coding region of the genome. This is because mutations on coding region may have a direct impact on the protein and may be implicated in a disease. The variants on non-coding region of the genome may also have impact but the challenge is that it is difficult to establish a testable hypothesis. Therefore, statistical methods were developed for variant prioritization by incorporating diverse functional evidence, so that variants with small effect sizes but possessing functional features may be prioritized over variants with similar effect sizes but less likely to be functional.

There are numerous variant annotation tools that attempt to associate variants to knowledge-based information and generate reports. The most commonly used tools include SIFT, SnpEff, Annovar, and VEP, which we will use to annotate the variants.

4.4.1 SIFT

The SIFT [11], which stands for Sorting Intolerant from Tolerant, was first introduced in 2001 as an online variant annotation tool that annotates coding region of genes with the missense variant effects on the translated protein. SIFT relies on the assumption that substitutions in conserved regions are more likely to be deleterious if the missense SNV

resulted in a nonsynonymous codon that is translated into an amino acid with different physicochemical properties. For instance, if a hydrophobic amino acid is replaced by another hydrophobic amino acid, SIFT will predict that change is tolerated; however, if it is substituted with a polar amino acid, the variant will be predicted as deleterious. SIFT algorithm avails of the NCBI PSI-BLAST as it uses the translated protein as a query sequence against a database of protein sequences. The search hit sequences are aligned using multiple sequence alignment (MSA) and the probabilities of all possible substitutions at each position are computed forming position-specific scoring matrix (PSSM), where each entry in the matrix represents the probability of observing an amino acid in that column of the alignment. The probabilities are normalized based on the consensus amino acids. Then, position with normalized probability ranges between 0 and 1. SIFT predicts that a SNV with a probability between 0.0 and 0.05 on that position is deleterious and will affect the function of the protein and a probability greater than 0.05 (>0.05) can be tolerated. SIFT also measures conservation of the sequence using the median sequence conservation, which ranges from 0 to $\log_2(20)$ or from 0 to 4.32, where median sequence conservation of 4.32 indicates that all sequences in the alignment are identical to each other, and hence, any variant in this region will be predicted as damaging. SIFT also reports the number of sequences at the variant position. The latest version of SIFT is SIFT 4G (SIFT for genomes), which is faster and enables practical computations on reference genomes using precomputed databases and also it provides SIFT prediction for more organisms. Hundreds of databases for different organisms are available.

Use the following steps to annotate variants using SIFT 4G on Linux terminal:

First, create a directory with the name of your choice or “sift4g” and change into it.

```
mkdir sift4g
cd sift4g
```

Open “<https://sift.bii.a-star.edu.sg/sift4g/public/>”. You will see databases of tens of organisms. Scroll down to the database of your interest, open its folder, and download the appropriate database build into your working directory. Since we have variants called above from human samples, we can download the latest human build GRCh38.78 by copying the link and using “wget” command and then unzip it using “unzip” or follow the instructions.

```
wget https://sift.bii.a-star.edu.sg/sift4g/public/Homo_sapiens/
GRCh38.78.zip
unzip GRCh38.78.zip
```

Each chromosome will have three files: a compressed file with “gz” file extension, a region file with “.region” file extension, and a chromosome statistics file with “.txt” file extension.

Download SIFT 4G Annotator Java executable file (.jar) in a directory or in your working directory:

```
wget https://github.com/paulineng/SIFT4G_Annotator/raw/master/
SIFT4G_Annotator.jar
```

Assuming that your VCF file is in the current directory, you can run the following on the command line:

```
java -jar SIFT4G_Annotator.jar \
-c -i humanSNP.vcf \
-d GRCh38.78 \
-r output -t
```

The option “-jar” for the SIFT 4G path, “-c” is essential for the command line, “-i” for the input path, “-d” for the database path, “-r” specifies the output folder, and “-t” is to extract annotations for multiple transcripts.

The above command generates two files: an Excel annotation file with “.xls” file extension and a VCF file.

Instead of the command line, you can also use SIFT 4G with a graphic user interface (GUI) by running the following command:

```
java -jar SIFT4G_Annotator.jar
```

This will open SIFT interface where you can browse to select the VCF file and the database and click Start to annotate the variants. The links of the two output files are given at the bottom of the GUI (Figure 4.9).

We can open the Excel file to check its content (Figure 4.10). We can notice that some annotations have been added to the variants, such as Ensemble transcript ID, gene ID, gene name, region, variant type (synonymous or nonsynonymous), SIFT score, median

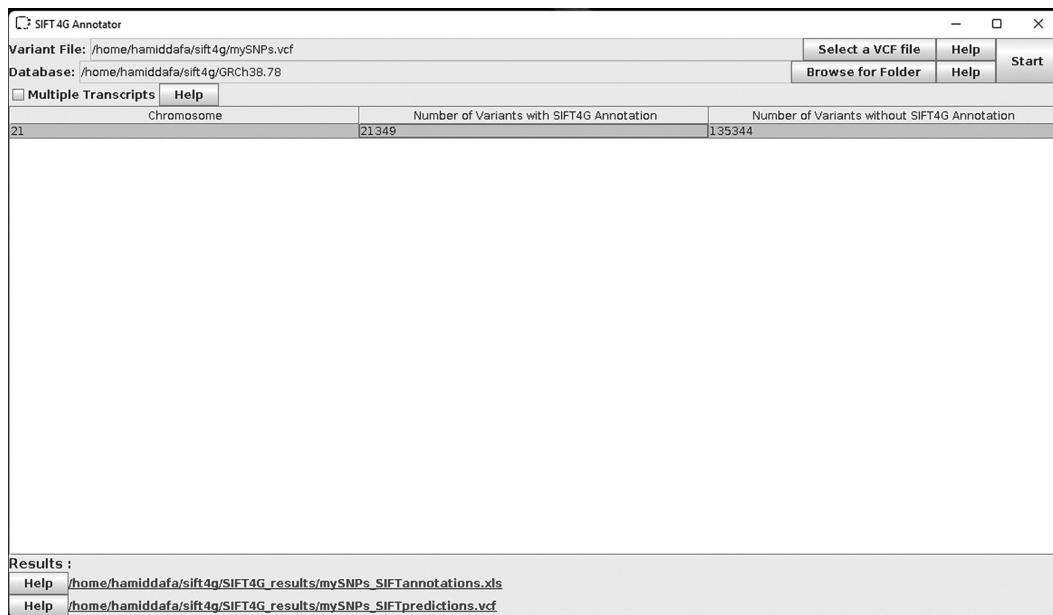


FIGURE 4.9 SIFT 4G annotator.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	CHROM	POS	REF_ALLELE	ALT_ALLELE	TRANSCRIPT_ID	GENE_ID	GENE_NAME	REGION	VARIANT_REF_AMINO	ALT_AMINO	AMINO_POS	SIFT_SCORE	SIFT_MEDIAN	NUM_SEQS	dbSNP	SIFT_PREDICTION
2	chr21	10542440	G	A	ENST0000022113	ENSG00000274391	TPTE	CDS	SYNONYM	A	37	0.909	3.19	18	r468525	TOLERATED
3	chr21	10542440	G	A	ENST0000027445	ENSG00000274391	TPTE	CDS	SYNONYM	A	37	0.923	3.16	22	r468525	TOLERATED
4	chr21	10542440	G	A	ENST0000018007	ENSG00000274391	TPTE	CDS	SYNONYM	A	37	1	4.32	11	r468525	TOLERATED
5	chr21	10569454	G	A	ENST0000022113	ENSG00000274391	TPTE	CDS	NONSNCN	R	195	0.005	3.05	41	r1810856	DELETERIOUS
6	chr21	10569454	G	A	ENST00000274391	ENSG00000274391	TPTE	CDS	NONSNCN	R	177	0.005	2.89	42	r1810856	DELETERIOUS
7	chr21	10569454	G	A	ENST0000027445	ENSG00000274391	TPTE	CDS	NONSNCN	R	157	0.006	2.89	42	r1810856	DELETERIOUS
8	chr21	10569454	G	A	ENST0000012746	ENSG00000274391	TPTE	CDS	NONSNCN	R	57	0.008	2.99	40	r1810856	DELETERIOUS
9	chr21	10569476	G	T	ENST0000027445	ENSG00000274391	TPTE	CDS	SYNONYM	L	164	1	2.89	42	r78225807	TOLERATED
10	chr21	10569476	G	T	ENST0000012746	ENSG00000274391	TPTE	CDS	SYNONYM	L	64	1	2.99	40	r78225807	TOLERATED
11	chr21	10569476	G	T	ENST0000018007	ENSG00000274391	TPTE	CDS	SYNONYM	L	202	1	3.05	41	r78225807	TOLERATED
12	chr21	10569476	G	T	ENST0000022113	ENSG00000274391	TPTE	CDS	SYNONYM	L	184	1	2.89	42	r78225807	TOLERATED
13	chr21	10569534	C	T	ENST0000027445	ENSG00000274391	TPTE	CDS	NONSNCN	R	184	0.008	2.6	65	r76723236	DELETERIOUS
14	chr21	10569534	C	T	ENST0000018007	ENSG00000274391	TPTE	CDS	NONSNCN	R	222	0.009	2.83	64	r76723236	DELETERIOUS
15	chr21	10569534	C	T	ENST0000012746	ENSG00000274391	TPTE	CDS	NONSNCN	R	204	0.009	2.62	66	r76723236	DELETERIOUS
16	chr21	10569534	C	T	ENST0000012746	ENSG00000274391	TPTE	CDS	NONSNCN	R	84	0.022	2.58	59	r76723236	DELETERIOUS
17	chr21	10577446	C	T	ENST0000027445	ENSG00000274391	TPTE	CDS	NONSNCN	R	231	0	2.55	65	r141045272	DELETERIOUS
18	chr21	10577446	C	T	ENST0000012746	ENSG00000274391	TPTE	CDS	NONSNCN	R	131	0	2.54	59	r141045272	DELETERIOUS
19	chr21	10577446	C	T	ENST0000018007	ENSG00000274391	TPTE	CDS	NONSNCN	R	269	0	2.78	69	r141045272	DELETERIOUS
20	chr21	10577446	C	T	ENST0000022113	ENSG00000274391	TPTE	CDS	NONSNCN	R	251	0	2.56	67	r141045272	DELETERIOUS
21	chr21	10578342	G	G	ENST0000012746	ENSG00000274391	TPTE	CDS	NONSNCN	R	150	0.016	2.54	58	r14933450	DELETERIOUS
22	chr21	10578342	G	G	ENST0000027445	ENSG00000274391	TPTE	CDS	NONSNCN	R	250	0.017	2.55	64	r14933450	DELETERIOUS
23	chr21	10578342	G	G	ENST0000022113	ENSG00000274391	TPTE	CDS	NONSNCN	R	270	0.03	2.57	67	r14933450	DELETERIOUS
24	chr21	10578342	G	G	ENST0000018007	ENSG00000274391	TPTE	CDS	NONSNCN	R	288	0.032	2.84	68	r14933450	DELETERIOUS

FIGURE 4.10 SIFT 4G annotation file.

conservation score, number of sequences, dbSNP accession, and SIFT prediction whether the variant is tolerated or deleterious.

4.4.2 SnpEff

SnpEff [12] is another variant annotation tool that categorizes the coding effects of variants based on their genomic locations such as introns, untranslated region (UTR), upstream, downstream, and splicing site. SnpEff predicts a variety of variant effects including synonymous or nonsynonymous substitution, start-gain codon, start-loss codon, stop-gain codon, stop-loss codon, or frameshifts.

In general, SnpEff consists of two main components: (i) database builds and (ii) variant effect calculation. The SnpEff database builds are usually distributed with SnpEff and there are around hundreds of databases available. A database build is a gzip-compressed serialized object that is formed of the genome FASTA sequence and an annotation file (in GTF or GFF format). These database files can be acquired from database resources such as ENSEMBL and UCSC. The variant effect calculation is performed after building the database. It begins with building a data structure which is a hash table interval trees indexed by chromosome. The data structure indexes intervals and makes their search efficient. The SnpEff program uses the VCF file as input and finds the intersections with the annotated database. The intersecting genomic regions are then identified and the variant effect is calculated from exonic region only. Simply, SnpEff will take information from the provided annotation database and populate the input VCF file by adding annotation into the INFO field name, ANN. Data fields are encoded separated by pipe sign "|"; and the order of fields is written in the VCF header. As examples, variants may be categorized by SnpEff as SNP (single-nucleotide polymorphism), Ins (insertion), Del (deletion), MNP (multiple-nucleotide polymorphism), or MIXED (multiple-nucleotide and InDel). The impacts of variants are classified into high, moderate, low, or modifier based on the affected region. A variant will have a high impact when it is disruptive and likely to cause protein truncation, loss of function, or triggering nonsense mediated decay. The variants with high impact are frameshift and stop-gain variants. The non-disruptive variants such as missense SNV and inframe deletion that might change protein effectiveness only are moderate impact

variants. The low-impact variants are the synonymous ones that do not change protein behavior; however, they may still have some effects. Variants on introns (intron-variants) and variants in the downstream region of a gene are annotated as MODIFIER since they affect non-coding regions, where prediction of the impact is difficult or there is no substantial evidence of any effect.

To use snpEff, we should install snpEff software and download the database of interest. Installation of SnpEff software requires Java V1.8 or later installed on your computer. The installation instructions are available at “<https://pcingola.github.io/SnpEff/download/>” or “<http://pcingola.github.io/SnpEff/>”. First, download the compressed folder using “wget” Linux command and then decompress it with “unzip” command. You can also install the binary.

```
wget https://snpeff.blob.core.windows.net/versions/snpEff_latest_core.zip
unzip snpEff_latest_core.zip
```

This will create the directory “snpEff”, which contains two Java executable files with “.jar” extension (snpEff.jar and snpSift.jar), a snpEff configuration file (snpEff.config), a license file, and four subdirectories (“examples”, “exec”, “galaxy”, and “script”). By default, the database will be stored in a subdirectory “data” in the “snpEff” directory but we can change it by opening “snpEff.config” and editing “data.dir=./data/” to the path that we need.

The database can be downloaded automatically and this is recommended but you can also install it manually. For example, if you need to download the human database manually, you can run the following:

```
java -jar snpEff.jar download GRCh38.
```

Figure 4.11 shows the directory structure of the snpEff. If you installed the binary files, you can use any of the executables without the “java” command as follows:

```
snpEff download GRCh38.
```

When you use snpEff, you must provide the right file path. For instance, if you are just one step out of the “snpEff” directory, then you can run:

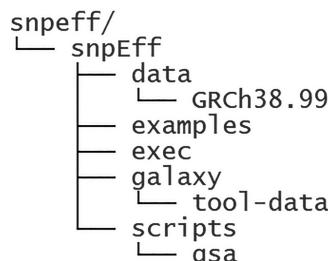


FIGURE 4.11 snpEff directory structure.

```
java -jar snpEff/snpEff.jar download GRCh38.99
```

To list all available SnpEff database, run the following command:

```
java -jar.snpEff/snpEff.jar databases
```

The “snpeff” is the current directory that we created to store the snpEff software, the VCF file, and the output. We will copy our VCF file to the root directory “snpeff”, while the snpEff executable file and database are in “snpEff”. After copying our VCF file “humanSNP.vcf” into the working directory, you can annotate it using the following command:

```
java -Xmx8g -jar snpEff/snpEff.jar GRCh38.99 humanSNP.vcf > mySNPanot.vcf
```

This command will produce three files: a VCF file (mySNPanot.vcf), gene file (snpEff_genes.txt), and summary file in html format (snpEff_summary.html). SnpEff adds functional annotations in the ANN keyword in the INFO field of the VCF output file. Figure 4.12 shows the VCF output file, which is modified to show ANN under INFO field. The INFO field may include the effect of the variant (stop loss, stop gain, etc.), effect impact on gene (High, Moderate, Low, or Modifier), or functional class of the variant (nonsense, missense, frameshift, etc.).

Moreover, we can view the summary on the html file to have a general idea about the type and regions and effects of the variants. If you have “firefox” installed, you can display the summary on the html file using the “firefox” command or you can open it with an Internet browser.

firefox.snpEff_summary.html

Figure 4.13 shows the summary of the annotation using SnpEff and variant rate details. Remember that the VCF file contains the variants of the human chromosome 21 only.

Figure 4.14 shows the number of variant effects by impact and by functional class. Only 68 SNVs (0.009%) have high impact. The remaining variants are SNV with moderate impact (0.149%), SNV with low impact (0.149), and modifier (99.575%).

#SnpEffVersion="5.1 (build 2022-01-21 06:23)", by Pablo Cingolan"											
#SnpEffCmd="SnpEff GRCh38.99_mySNPs.vcf"											
#INFO<=D;ANN;Number=.,Type=String,Description="Functional annotations: 'Allele Annotation Annotation_Impact Gene_Name Gene_ID Feature_Type Feature_ID Transcript_BioType'"											
#INFO<=D;DP;Number=.,Type=String,Description="Predicted loss of function effects for this variant. Format: 'Gene_Name Gene_ID Number_of_transcripts_in_gene Percent_of_transcripts_in_gene'"											
#INFO<=D;NMD;Number=.,Type=String,Description="Predicted nonsense mediated decay effects for this variant. Format: 'Gene_Name Gene_ID Number_of_transcripts_in_gene Percent_of_transcripts_in_gene'"											
CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	ERR1019034	ERR1019044	ERR1019045
chr21	5033884.	G	A	231.23	PASS	ANN A intron_variant MODIFIER FP565260.3-ENST GT:DP:GQ:PL 0/0:8.24:24.0/0:11.13;10.29 0:0.27:27.69 0:0.25:25.06					
chr21	5035658.	C	T	5536.83	PASS	ANN T 3_prime_UTR_variant MODIFIER FP565260.7-ENST GT:DP:GQ:PL 1/0:11.11;33.0/0:12.10;22.99 1/0:14.44;49.95					
chr21	5038313.	C	T	61.6	PASS	ANN T downstream_gene_variant MODIFIER FP565 GT:DP:GQ:PL 0/0:0.00:0.0/0:1.33;6.68:6.61 0/0:1.3-3.0/3.0 0/0:6.06-3.0/3.0					
chr21	5075887.	A	C	1666.97	PASS	ANN C intergenic_region MODIFIER FP565260.3-3.FP GT:DP:GQ:PL 1/0:11.11;33.0/0:9.0:9.24 0/0:0.6;18.0:1.0 0/0:0.00:0.0/0:0.00					
chr21	5097442.	C	T	77.64	PASS	ANN T intron_variant MODIFIER GATDB38 ENSG000 GT:DP:GQ:PL 0/0:0.00:0.0/0:1/0:1.33;8.9 0/0:1.1;3.0/3.0 0/0:2.2;6.0/6.0					
chr21	1.1E+07.	G	A	4234.94	PASS	ANN A synonymous_variant MODERATE TPT ENSG000 GT:DP:GQ:PL 0/1:10.65;7.99 0/0:1.55;12.67;9.0/0:1.57;27.77 0/0:1.57;16.73					
chr21	1.1E+07.	G	A	1970.6	PASS	ANN A missense_variant MODERATE TPT ENSG000 GT:DP:GQ:PL 0/0:6.66;6.69 0/0:16.11;22.83 0/0:1.1;3.0/3.0					
chr21	1.1E+07.	G	A	457.61	PASS	ANN A missense_variant MODERATE TPT ENSG00 GT:DP:GQ:PL 0/0:6.66;6.69 0/0:55.05;55.99 0/0:46.04;46.99 0/0:16.11;28.79					
chr21	1.1E+07.	A	C	4322.94	PASS	ANN C splice_region_variant LOW TPT ENSG000 GT:DP:GQ:PL 0/1:6.21;10.72;9.0 0/0:16.10;13.73 0/0:5.1;9.60;99/0:17.10;15.75;5.0					
chr21	1.1E+07.	T	G	2481.2	PASS	ANN G 3_prime_UTR_variant MODIFIER TPT ENSG00 GT:DP:GQ:PL 0/0:40.40;40.0 0/0:1.56;13.69 0/0:35.35;99/0:1/16.76;14.90					

FIGURE 4.12 A VCF annotated with SnpEff.

Summary	
Genome	GRCh38.99
Date	2022-01-29 10:05
SnpEff version	SnpEff 5.1 (build 2022-01-21 06:23), by Pablo Cingolani
Command line arguments	SnpEff GRCh38.99 mySNPs.vcf
Warnings	47,561
Errors	0
Number of lines (input file)	156,693
Number of variants (before filter)	158,773
Number of not variants (i.e. reference equals alternative)	0
Number of variants processed (i.e. after filter and non-variants)	157,119
Number of known variants (i.e. non-empty ID)	0 (0%)
Number of multi-allelic VCF entries (i.e. more than two alleles)	2,069
Number of effects	740,232
Genome total length	63,147,197,748
Genome effective length	46,709,983
Variant rate	1 variant every 297 bases

Variants rate details			
Chromosome	Length	Variants	Variants rate
21	46,709,983	157,119	297
Total	46,709,983	157,119	297

FIGURE 4.13 SnpEff annotation summary.

Type (alphabetical order)	Count	Percent
HIGH	68	0.009%
LOW	1,975	0.267%
MODERATE	1,106	0.149%
MODIFIER	737,083	99.575%

Type (alphabetical order)	Count	Percent
MISSENSE	1,109	45.507%
NONSENSE	19	0.78%
SILENT	1,309	53.714%

FIGURE 4.14 Variant effect summary.

Figure 4.15 shows the number of effects by type and region. Figure 4.16 shows a bar chart showing the percentage of effects by region.

4.3.3 ANNOVAR

ANNOVAR [13] is one of the most commonly used annotation tools that is used to annotate SNVs and InDels with different types of annotations including functional consequence on genes, inferring cytogenetic bands, functional importance, impacts of variants in conserved regions, and identifying variants reported in the NCBI dbSNP and the 1000

Type	Region				
Type (alphabetical order)	Count	Percent	Type (alphabetical order)	Count	Percent
3_prime_UTR_variant	3,308	0.447%	DOWNSTREAM	56,841	7.679%
5_prime_UTR_premature_start_codon_gain_variant	108	0.015%	EXON	10,670	1.441%
5_prime_UTR_variant	750	0.101%	INTERGENIC	68,013	9.188%
downstream_gene_variant	56,841	7.672%	INTRON	545,141	73.645%
intergenic_region	68,013	9.18%	SPLICE_SITE_ACCEPTOR	32	0.004%
intragenic_variant	1	0%	SPLICE_SITE_DONOR	14	0.002%
intron_variant	545,570	73.641%	SPLICE_SITE_REGION	571	0.077%
missense_variant	1,106	0.149%	TRANSCRIPT	1	0%
non_coding_transcript_exon_variant	8,403	1.134%	UPSTREAM	54,783	7.401%
splice_acceptor_variant	32	0.004%	UTR_3_PRIME	3,308	0.447%
splice_donor_variant	14	0.002%	UTR_5_PRIME	858	0.116%
splice_region_variant	592	0.08%			
start_lost	1	0%			
stop_gained	19	0.003%			
stop_lost	2	0%			
synonymous_variant	1,309	0.177%			
upstream_gene_variant	54,783	7.395%			

FIGURE 4.15 Number of variant effects by type and region.

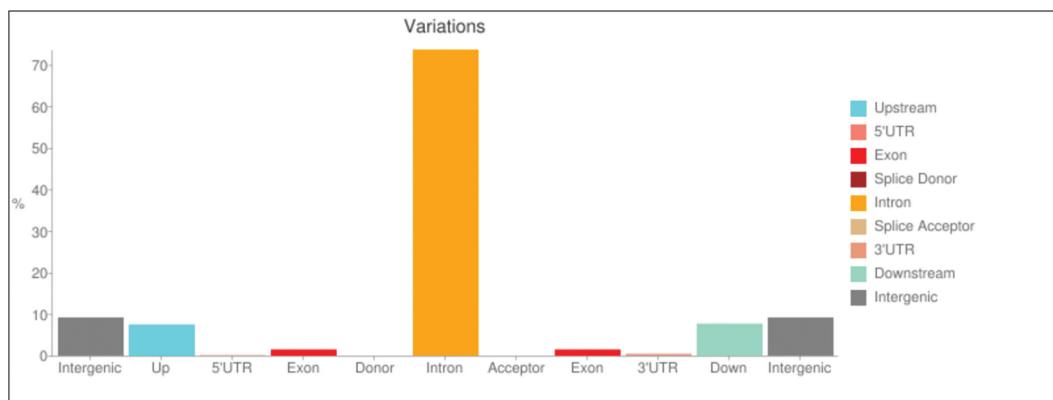


FIGURE 4.16 A bar chart shows percentage of variants by region.

Genomes Project. In addition to variant annotation with respect to genes, ANNOVAR has the ability to perform annotation based on genomic region and to compare variants to existing variation databases. In general, the types of annotations with ANNOVAR can be grouped into the following: (i) gene-based annotation which identifies the effects of variants on the proteins, (ii) region-based annotation identifies the affected region (e.g., conserved region), and (iii) filter-based annotation identifies variants based on a specific database such as dbSNP, ExAC, 1000 Genome Project, and gnomAD. The filter-based annotation may also generate scores including SIFT, PolyPhen, LRT, MutationTaster, MutationAssessor, FATHMM, MetaSVM, and MetaLR.

ANNOVAR uses annotation databases to perform the above types of annotation. The annotation databases are built with the organism annotation file in GFF3 format.

TABLE 4.4 ANNOVAR Script Files

ANNOVAR Program	Description
annotate_variation.pl	The core ANNOVAR program for annotation and database download
coding_change.pl	To calculate the mutated sequence and make inference
convert2annovar.pl	To convert genotype-calling file format into ANNOVAR input format
retrieve_seq_from_fasta.pl	To retrieve genomic nucleotide, cDNA sequences, or translated amino acid sequences from FASTA file
table_annovar.pl	To generate a tab-delimited output file with annotation columns
variants_reduction.pl	For prioritizing causal variants

Ready-built human databases are available at ANNOVAR server, UCSC Genome Browser website, or third parties and can be downloaded using “annotate_variation.pl” program. For the non-human species which have no available annotation databases, a database can be built from the FASTA sequence of the reference genome and the GFF/GTF annotation file of that species. Those two files can be downloaded from databases such as NCBI Genome database or UCSC database.

As shown in Table 4.4, ANNOVAR consists of six Perl files that can be used as command-line programs on any computer with Perl installed. The download instructions are available at “<https://annovar.openbioinformatics.org/en/latest/user-guide/download/>”. You may be asked to register with your school email. The download link will be emailed to you, and then you can download the compressed file onto your computer and decompress it with “tar xvf” command. If you are using Linux, you can add ANNOVAR to the path by adding the following line to the end of “.bashrc” file:

Export PATH=“YOURPATH/annovar:\$PATH”

4.3.3.1 Annotation Databases

For variant annotation, ANNOVAR uses annotation databases of an organism to be downloaded in a directory. Databases can be downloaded from UCSC Genome Browser, 1000 genome project or ANNOVAR website, or from a third-party URL. You can use “annotate_variation.pl” to annotate, download a database, or list the available databases for a specific build. The general syntax is as follows:

```
annotate_variation.pl \
[arguments] \
<query-file|table-name> \
<database-location>
```

For the complete list of argument run:

```
annotate_variation.pl -h
```

To list the available annotation databases for the hg19 build of the human reference genome, you can run the following command:

```
annotate_variation.pl -webfrom annovar -downdb avdblist -buildver
hg19 dblist
```

The list of the databases of the hg19 build will be saved in a file “hg19_avdblist.txt” in “dblist” subdirectory. You can open that file and study the available annotation databases.

The “-webfrom” argument specifies the source of database; the values can be “ucsc”, “annovar”, or a URL where the database is available.

The “-downdb” argument is the command to download an annotation database from the source specified by “-webfrom”.

The “-buildver” argument specifies the genome build version (e.g., hg19).

For example, the following command downloads some annotation databases for hg19 build and they will be stored in “humandb” directory:

To download the gene-based annotation database for human variants:

```
annotate_variation.pl \
-buildver hg19 \
-downdb \
-webfrom annovar refGene humandb/
```

To download region-based annotation (cytogenetic bands) database of the human genome:

```
annotate_variation.pl \
-buildver hg19 \
-downdb cytoBand humandb/
```

To download filter-based ExAC (Exome Aggregation Consortium) annotation database:

```
annotate_variation.pl \
-buildver hg19 \
-downdb \
-webfrom annovar exac03 humandb/
```

To download the filter-based NCBI dbSNP annotation database (version 147):

```
annotate_variation.pl \
-buildver hg19 \
-downdb \
-webfrom annovar avsnp147 humandb/
```

To download the filter-based dbNSFP annotation database, which was developed for functional prediction and annotation of all potential nonsynonymous SNVs in the human genome:

```
annotate_variation.pl \
-buildver hg19 \
```

```
-downdb \
-webfrom annovar dbnsfp30a humandb/
```

The database files are downloaded into the specified directory “humandb”. Save the annotation databases of each organism in a separate file.

Not all non-human organisms have annotation databases. In this case, you can build an annotation database for any organism by yourself. The following steps show how to build a gene-based annotation database. As an example, we will build an annotation database for SARS-CoV-2 and we will use it later to annotate the variants called in a previous example. The following are the steps to build SARS-CoV-2 gene-based annotation database:

1. Download the reference genome sequence of the organism in FASTA format and the sequence annotation file in GFF/GTF format. For SARS-CoV-2, we can download both files from the NCBI Genome database at

```
https://www.ncbi.nlm.nih.gov/genome/86693?genome\_assembly\_id=757732
```

Use the following commands to create a directory “sarscov2db” and download the reference FASTA file and GFF file into it:

```
mkdir sarscov2db
cd sarscov2db
wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/GCF\_009858895.2\_ASM985889v3/GCF\_009858895.2\_ASM985889v3\_genomic.fna.gz
wget https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/009/858/895/GCF\_009858895.2\_ASM985889v3/GCF\_009858895.2\_ASM985889v3\_genomic.gff.gz
```

Then, decompress the two files with “gunzip” command:

```
gunzip GCF_009858895.2_ASM985889v3_genomic.fna.gz
gunzip GCF_009858895.2_ASM985889v3_genomic.gff.gz
```

2. Use the “gff3ToGenePred” tool to convert the GFF file to GenePred file, which is a file format used to specify the gene track annotations for an imported genome. For GTF format, use “gtfToGenePred” to convert it into GenePred file. Both “gff3ToGenePred” and “gtfToGenePred” are ones of the UCSC Genome Browser application binaries built for standalone command-line use on Linux and UNIX platforms. They can be downloaded by choosing the right platform at “<http://hgdownload.soe.ucsc.edu/admin/exe/>”. For the sake of simplicity, we can download “gff3ToGenePred” in the same “sarscov2db” directory and use “chmod” to allow it to run as a program:

```
wget http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86\_64/gff3ToGenePred
chmod +x gff3ToGenePred
```

If you wish to download all UCSC Genome Browser binaries, run the following:

```
mkdir ucsc
cd ucsc
rsync -aP rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.
x86_64/ ./
```

This will download all binaries in “ucsc” directory.

After downloading the right tool, you can use it to convert the annotation GFF file into GenePred file:

```
gff3ToGenePred \
  GCF_009858895.2_ASM985889v3_genomic.gff \
  SARSCOV2_refGene.txt
```

This will create “SARSCOV2_refGene.txt”, which is a GenePred file.

3. Use ANNOVAR “retrieve_seq_from_fasta.pl” script to generate a transcript FASTA file from the reference sequence.

```
retrieve_seq_from_fasta.pl \
  --format refGene \
  --seqfile GCF_009858895.2_ASM985889v3_genomic.fna \
  SARSCOV2_refGene.txt \
  --out SARSCOV2_refGeneMrna.fa
```

This will create a transcript FASTA file “SARSCOV2_refGeneMrna.fa”.

Thus, the gene-based database for SARS-CoV-2 variants is ready to use. We will use it in a later example.

4.3.3.2 ANNOVAR Input Files

The “annotate_variation.pl” script is the core ANNOVAR program for variant annotation. The raw variants (SNVs or InDels) must be in an ANNOVAR input file for annotate_variation.pl. The ANNOVAR input file is a plain text file that contains space- or tab-delimited five columns for chromosome, start position, end position, the reference nucleotides, and the observed nucleotides. Additional columns can be added. You can open the example ANNOVAR input file “ex1.avinput” in the “example” directory to have an idea about how it looks. Use “less -S ex1.avinput” to display the file.

Since variants come in different variant calling file formats, “convert2annovar.pl” script can be used to convert those files to the ANNOVAR input file format. The variant calling files that can be converted by that program include VCF format, samtools genotype-calling pileup format, Illumina export format from GenomeStudio, SOLiD GFF genotype-calling format, and complete genomics variant format. To learn about the use and options of “convert2annovar.pl” script, run the following:

```
convert2annovar.pl -h
```

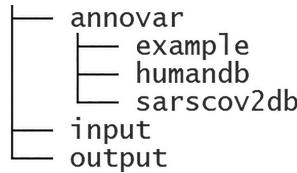


FIGURE 4.17 The directory tree of the ANNOVAR.

chr21	5033884	5033884	G	A	0.03846	231.23	18	chr21	5033884	.	G	A	231.23	PA>
chr21	5035658	5035658	C	T	0.5385	5536.83	23	chr21	5035658	.	C	T	5536.83	PA>
chr21	5038298	5038298	A	G	0.1923	270.10	7	chr21	5038298	.	A	G	270.10	VO>
chr21	5038313	5038313	C	T	0.03846	61.60	0	chr21	5038313	.	C	T	61.60	PA>
chr21	5049383	5049383	G	A	0.03846	40.92	0	chr21	5049383	.	G	A	40.92	VO>
chr21	5057887	5057887	A	C	0.4615	1666.97	0	chr21	5057887	.	A	C	1666.97	PA>
chr21	5064036	5064036	G	A	0.07692	69.25	0	chr21	5064036	.	G	A	69.25	VO>
chr21	5097442	5097442	C	T	0.07692	77.64	0	chr21	5097442	.	C	T	77.64	PA>
chr21	5097921	5097921	C	T	0.1923	988.62	6	chr21	5097921	.	C	T	988.62	VO>
chr21	5114338	5114338	T	C	0.03846	42.82	0	chr21	5114338	.	T	C	42.82	PA>
chr21	5134181	5134181	C	T	0.03846	32.53	0	chr21	5134181	.	C	T	32.53	VO>
chr21	5289835	5289835	C	T	0.07692	81.75	1	chr21	5289835	.	C	T	81.75	VO>
chr21	5289851	5289851	A	T	0.07692	81.31	1	chr21	5289851	.	A	T	81.31	VO>
chr21	5292724	5292724	C	A	0.07692	57.26	0	chr21	5292724	.	C	A	57.26	PA>
chr21	5298170	5298170	C	T	0.07692	107.68	0	chr21	5298170	.	C	T	107.68	VO>
chr21	5306986	5306986	T	G	0.07692	57.94	0	chr21	5306986	.	T	G	57.94	VO>
chr21	5312404	5312404	A	G	0.03846	76.61	0	chr21	5312404	.	A	G	76.61	PA>
chr21	5312605	5312605	T	G	0.07692	156.80	3	chr21	5312605	.	T	G	156.80	PA>
chr21	5312968	5312968	G	A	0.07692	126.63	4	chr21	5312968	.	G	A	126.63	PA>

FIGURE 4.18 ANNOVAR input file.

The VCF file format is the standard format for variant calling. The VCF file can be converted into ANNOVAR input file by using “-format vcf4” argument.

Figure 4.17 shows the directory tree which includes “annovar” directory that contains the ANNOVAR scripts and subdirectories and the “input” directory that includes the VCF files (sarscov2.vcf and humanSNP.vcf) from the previous SARS-CoV-2 and human variant calling examples. We copied them to this directory for simplicity. The following command will convert “humanSNP.vcf” file into ANNOVAR input format “humanSNP.avinput”:

```
convert2annovar.pl \
  -format vcf4 input/humanSNP.vcf \
> input/humanSNP.avinput
```

Figure 4.18 shows the ANNOVAR input file, which includes the first five essential columns and additional three columns.

For converting other variant calling file formats, run “convert2annovar.pl -h”. This command is also used with “-dbSNP” option to add the dbSNP accessions.

Variant annotation with ANNOVAR:

The “annotate_variation.pl” script is the core program for ANNOVAR annotation. It requires ANNOVAR input file. However, “table_annovar.pl” script is also used for annotation and it takes a VCF file as input.

```
./annotate_variation.pl \
  -out ../output/humanSNPannot \
```

```
-build hg19 \
./input/humanSNP.avininput \
humandb/
```

The above command generates three files with the “humanSNPanno” prefix as shown in Figure 4.19. The “humanSNPanno.variant_function” file contains annotation for all variants, by adding two columns to the beginning of each input line (Figure 4.20).

The first column is annotated with the affected part of the gene. The parts can be exonic, splicing, nrRNA, UTR5, UTR3, intronic, upstream, downstream, or intergenic region. The second added column annotates the gene name or names.

The second annotation output file, “humanSNPanno.exonic_variant_function”, contains the amino acid changes as a result of the exonic variant (Figure 4.21).

```
|- humanSNP2.avininput
|- humanSNP2.hg19_multianno.txt
|- humanSNP2.hg19_multianno.vcf
|- humanSNPanno.exonic_variant_function
|- humanSNPanno.log
|- humanSNPanno.variant_function
```

FIGURE 4.19 Variant annotation files.

intergenic	LINC02246(dist=39036),NRIP1(dist=3531)	chr21	16330025	16330025	C	T	ho		
intergenic	LINC02246(dist=39706),NRIP1(dist=2861)	chr21	16330695	16330695	C	T	ho		
intergenic	LINC02246(dist=40459),NRIP1(dist=2108)	chr21	16331448	16331448	A	G	ho		
downstream	NRIP1(dist=101),chr21	16333455	16333455	G	A	hom	17895.94	35	
UTR3	NRIP1(NM_003489:c.*2518C>G)	chr21	16334519	16334519	G	C	hom	14296.25	
UTR3	NRIP1(NM_003489:c.*1636G>A)	chr21	16335401	16335401	C	T	hom	6445.02	8
UTR3	NRIP1(NM_003489:c.*1626T>A)	chr21	16335411	16335411	A	T	hom	6050.02	8
UTR3	NRIP1(NM_003489:c.*385A>G)	chr21	16336652	16336652	T	C	het	11581.08	
UTR3	NRIP1(NM_003489:c.*312C>G)	chr21	16336725	16336725	G	C	het	12415.08	
exonic	NRIP1 chr21	16337059	16337059	T	A	het	10937.08	24	
exonic	NRIP1 chr21	16337324	16337324	T	A	het	12879.08	37	
exonic	NRIP1 chr21	16337680	16337680	A	G	het	565.61	34	
exonic	NRIP1 chr21	16338103	16338103	G	A	het	13530.08	32	
exonic	NRIP1 chr21	16338393	16338393	T	C	het	11010.08	37	
exonic	NRIP1 chr21	16338549	16338549	C	T	het	2267.96	42	
exonic	NRIP1 chr21	16338745	16338745	T	C	het	1538.83	31	
UTR5	NRIP1(NM_003489:c.-281C>A)	chr21	16340794	16340794	G	T	hom	14784.04	
UTR5	NRIP1(NM_003489:c.-312C>A)	chr21	16340825	16340825	G	T	het	851.45	31
intronic	NRIP1 chr21	16340876	16340876	T	C	het	10836.08	29	
intronic	NRIP1 chr21	16341415	16341415	C	T	het	14101.08	38	

FIGURE 4.20 The ANNOVAR variant annotation file.

line8519	non synonymous SNV	POTED:NM_174981:exon10:c.C1484G:p.T495S	chr21	15011910	15
line9062	non synonymous SNV	RBM11:NM_144770:exon2:c.A231c:p.G77G,RBM11:NM_001320602:exon2:c.A231c:p.G77G	chr21		ch
line9275	non synonymous SNV	SAMSN1:NM_001256370:exon9:c.G1198T:p.D400Y,SAMSN1:NM_001286523:exon9:c.G78	chr21		
line9298	non synonymous SNV	SAMSN1:NM_001256370:exon3:c.G277T:p.G93W,SAMSN1:NM_022136:exon2:c.G73T:p.G	chr21		
line9359	non synonymous SNV	SAMSN1:NM_001256370:exon2:c.T141A:p.P47P,	chr21	15954577	15954577
line9817	non synonymous SNV	NRIP1:NM_003489:exon4:c.C3455T:p.T1152M,	chr21	16337059	16
line9818	non synonymous SNV	NRIP1:NM_003489:exon4:c.A3190T:p.T1064S,	chr21	16337324	16
line9819	non synonymous SNV	NRIP1:NM_003489:exon4:c.G2834C:p.C945S,	chr21	16337680	16337680
line9820	non synonymous SNV	NRIP1:NM_003489:exon4:c.A2411T:p.E804V,	chr21	16338103	16338103
line9821	non synonymous SNV	NRIP1:NM_003489:exon4:c.T2121G:p.L707L,	chr21	16338393	T
line9822	non synonymous SNV	NRIP1:NM_003489:exon4:c.A1965A:p.L655L,	chr21	16338549	16338549
line9823	non synonymous SNV	NRIP1:NM_003489:exon4:c.A1769G:p.K590R,	chr21	16338745	16338745
line13553	non synonymous SNV	CXADR:NM_001207063:exon3:c.G301G:p.G101G,CXADR:NM_001207066:exon3:c.G301G:p.G101G,	chr21		
line13554	non synonymous SNV	CXADR:NM_001207063:exon3:c.G406T:p.V136L,CXADR:NM_001207066:exon3:c.G406T:	chr21		
line13588	non synonymous SNV	CXADR:NM_001207063:exon4:c.T507T:p.L169L,CXADR:NM_001207066:exon4:c.T507T:p.L169L,	chr21		
line14003	non synonymous SNV	C21orf91:NM_017447:exon3:c.G470T:p.C157F,C21orf91:NM_001100421:exon3:c.G47>	chr21		
line14004	non synonymous SNV	C21orf91:NM_017447:exon3:c.A413T:p.Q138L,C21orf91:NM_001100421:exon3:c.A413	chr21		

FIGURE 4.21 Exonic variant function file.

The first column includes the line number in the original input file. The second column shows the functional consequences of the variant. The possible consequences are nonsynonymous SNV, synonymous SNV, frameshift insertion, frameshift deletion, nonframeshift insertion, nonframeshift deletion, frameshift block substitution, or nonframeshift block substitution. The third column includes the gene name, the transcript identifier, and the sequence change in the corresponding transcript.

The “annotate_variation.pl” tool has numerous arguments. Use “annotate_variation.pl -h” to display the complete list of arguments.

ANNOVAR provides “table_annovar.pl” script as an easy way to annotate variants in a VCF file as an input. No need to convert VCF file into ANNOVAR input file. It takes a VCF file as an input and generates a tab-delimited output file with many columns, each represents one set of annotations. It also generates a new output VCF file with the INFO field filled with annotation information.

```
./table_annovar.pl ../input/humanSNP.vcf humandb/ \
-buildver hg19 \
-out ../output/humanSNP2 \
-remove \
-protocol refGene,cytoBand,exac03,avsnp147,dbnsfp30a \
-operation g,r,f,f,f \
-nastring . \
-vcfinput
```

The “-remove” option removes all temporary files. The “-protocol” option is comma-delimited string that specifies an annotation protocol. These strings typically represent database names in ANNOVAR. The “-operation” option tells ANNOVAR which operations to use for each of the protocols, where “g” means gene-based, “gx” means gene-based with cross-reference annotation (from -xref argument), “r” means region-based, and “f” means filter-based. The above ANNOVAR command generated three output files: “humanSNP2.avinput”, “humanSNP2.hg19_multianno.txt”, and “humanSNP2.hg19_multianno.vcf”. The first one is an ANNOVAR input file. The second one is the annotation file with annotation columns, and the last one is a VCF file with annotation added to INFO fields. Open each of these files and study their contents.

We can also try the annotation database that we created for SARS-CoV-2. We can annotate “sarscov2.vcf”, which was generated from a previous variant calling example. You can copy it to the “input” directory for easy use. Thus, we can annotate it using the following script:

```
./table_annovar.pl ../input/sarscov2.vcf sarscov2db/ \
-buildver SARS202 \
-out ../output/cov2SNP \
-remove \
-protocol refGene \
```

exonic	gene-GU280_gp02	NC_045512.2	24410	24410	G	A	0.5	556	1004	NC_045512.2	24>
exonic	gene-GU280_gp02	NC_045512.2	24672	24672	C	T	0.5	55	1000	NC_045512.2	24>
exonic	gene-GU280_gp02	NC_045512.2	24783	24783	A	G	0.125	98	1033	NC_045512.2	24>
exonic	gene-GU280_gp02	NC_045512.2	24944	24944	G	C	0.75	333	1004	NC_045512.2	24>
exonic	gene-GU280_gp02	NC_045512.2	25194	25194	A	G	0.125	5.94522	39	NC_045512.2	25>
exonic	gene-GU280_gp02	NC_045512.2	25207	25207	C	T	0.125	5.95173	39	NC_045512.2	25>
exonic	gene-GU280_gp02	NC_045512.2	25352	25352	G	T	0.375	540	987	NC_045512.2	25>
exonic	gene-GU280_gp03	NC_045512.2	25469	25469	C	T	0.5	464	35	NC_045512.2	25>
exonic	gene-GU280_gp03	NC_045512.2	25567	25567	G	A	0.125	11.9395	39	NC_045512.2	25>
exonic	gene-GU280_gp03	NC_045512.2	25622	25622	T	C	0.5	91	1000	NC_045512.2	25>
exonic	gene-GU280_gp03	NC_045512.2	25702	25702	C	T	0.25	399	984	NC_045512.2	25>
exonic	gene-GU280_gp03	NC_045512.2	26144	26144	G	C	0.5	83	1000	NC_045512.2	26>
exonic	gene-GU280_gp04	NC_045512.2	26458	26458	G	T	0.5	936	1002	NC_045512.2	26>
upstream;downstream	gene-GU280_gp05	gene-GU280_gp06	gene-GU280_gp07	(dist=31)	gene-GU280_gp03	gene-GU280_gp04	(dist=31)	gene-GU280_gp05	gene-GU280_gp06	gene-GU280_gp07	(dist=31)
exonic	gene-GU280_gp05	NC_045512.2	26657	26657	T	C	0.125	9.80815	11	NC_045512.2	26>
exonic	gene-GU280_gp05	NC_045512.2	26664	26664	A	C	0.5	114	1004	NC_045512.2	26>

FIGURE 4.22 All-variant annotation file of SARS-CoV-2.

```
-operation g \
-nastring . \
-vcfinput
```

The all-variant annotations of SARS-CoV-2 are shown as in Figure 4.22.

4.5 SUMMARY

The high-throughput sequencing makes variant discovery much easier than the use of traditional methods like microarrays. Raw data obtained from sequencing technology is used for the detection of variants including base substitutions, insertions, deletions, and structural variants. Variants can be in any region of the genome; however, only variants that affect functions of the genes are studied. The consequences of variants depend on the affected regions and they may be deleterious implicating in healthy conditions and disease like cancers or may lead to the appearance of a new strain in bacteria and viruses that is more infectious and lethal like the recent variants of SARS-CoV2 or more antibiotic-resistant strain of bacteria. This why the variant discovery using sequencing data gained importance and it is widely used in genetics, medical diagnosis, and drug discovery.

Sequencing depth, paired-end sequencing, and the use of long reads make variant detection more accurate and allow detection of large-scale variants like structural variants, insertions, and deletions.

The variant calling pipelines use SAM/BAM files of whole genome, whole transcriptome, or targeted gene sequences to discover the bases in the samples that are different from the bases on the same locations on the reference genome. Variant calling programs use two approaches for variant calling. The first approach is used by bcftools and it is based on consensus sequence which is formed by collapsing the piled-up aligned reads. The second approach is used by the recent variant callers like GATK. This approach is based on haplotypes of the variants that are more likely to be inherited together. GATK 4 is the most commonly used program for variant calling. It uses an advanced workflow pipeline called GATK best practice pipeline which leads to the detection of accurate variants.

After variant identification with a variant calling program and filtering, variants can be annotated by assigning functional information to variants using annotation programs.

REFERENCES

1. Monroe JG, Srikant T, Carbonell-Bejerano P, Becker C, Lensink M, Exposito-Alonso M, Klein M, Hildebrandt J, Neumann M, Kliebenstein D et al: Mutation bias reflects natural selection in *Arabidopsis thaliana*. *Nature* 2022, 602(7895): 101–105.
2. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST et al: The variant call format and VCFtools. *Bioinformatics* 2011, 27(15):2156–2158.
3. Tam V, Patel N, Turcotte M, Bossé Y, Paré G, Meyre D: Benefits and limitations of genome-wide association studies. *Nat Rev Genetics* 2019, 20(8):467–484.
4. Martincorena I, Campbell PJ: Somatic mutation in cancer and normal cells. *Science (New York, NY)* 2015, 349(6255):1483–1489.
5. Population genetics [<https://www.nature.com/subjects/population-genetics>]
6. Danecek P, Bonfield JK, Liddle J, Marshall J, Ohan V, Pollard MO, Whitwham A, Keane T, McCarthy SA, Davies RM et al: Twelve years of SAMtools and BCFtools. *Gigascience* 2021, 10(2).
7. Pegueroles C, Mixão V, Carreté L, Molina M, Gabaldón T: HaploTypo: a variant-calling pipeline for phased genomes. *Bioinformatics* 2020, 36(8): 2569–2571.
8. Yun T, Li H, Chang PC, Lin MF, Carroll A, McLean CY: Accurate, scalable cohort variant calls using DeepVariant and GLnexus. *Bioinformatics* 2021, 36(24): 5582–5589.
9. Rivas MA, Graham D, Sulem P, Stevens C, Desch AN, Goyette P, Gudbjartsson D, Jónsdóttir I, Thorsteinsdóttir U, Degenhardt F et al: A protein-truncating R179X variant in RNF186 confers protection against ulcerative colitis. *Nat Commun* 2016, 7:12342.
10. Stitzel NO, Stirrups KE, Masca NG, Erdmann J, Ferrario PG, König IR, Weeke PE, Webb TR, Auer PL, Schick UM et al: Coding Variation in ANGPTL4, LPL, and SVEP1 and the Risk of Coronary Disease. *N Engl J Med* 2016, 374(12):1134–1144.
11. Ng PC, Henikoff S: SIFT: Predicting amino acid changes that affect protein function. *Nucleic Acids Res* 2003, 31(13):3812–3814.
12. Cingolani P, Platts A, Wang le L, Coon M, Nguyen T, Wang L, Land SJ, Lu X, Ruden DM: A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of *Drosophila melanogaster* strain w1118; iso-2; iso-3. *Fly (Austin)* 2012, 6(2):80–92.
13. Yang H, Wang K: Genomic variant annotation and prioritization with ANNOVAR and wANNOVAR. *Nat Protoc* 2015, 10(10):1556–1566.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

RNA-Seq Data Analysis

5.1 INTRODUCTION TO RNA-SEQ

RNA sequencing or shortly RNA-Seq is a high-throughput sequencing technique to examine gene expression or profiling in biological samples. Rather than the whole genome, RNA-Seq focuses only on transcriptomes to tell us which one of the genes in the samples are turned on or off and to what extent. The transcriptome is the set of all messenger RNA transcribed from genes in cells. Genes represent a small portion of the whole genome of an organism. For instance, genes are only around 3% of the human genome. RNA, in general, can either be translated into proteins or play functional role such as rRNA for protein synthesis, tRNA for amino acid transfer, and microRNA that plays a role in gene regulation. Out of whole transcriptome, protein-coding RNA (mRNA) is around 2%–4%, while the greatest percentage of RNA molecules is the other types of RNA. The gene expression studies mainly focus on mRNA only because it is translated into proteins. Proteins are required for the structure, function, and regulation of the cells forming body's tissues and organs. Dysfunctions of proteins implicate in most of the diseases and healthy conditions. By studying the mRNA, we can find out which genes are active in a particular cell type or under a certain condition, giving us a clue about the function of the cells and the biological activities under that condition. Moreover, we can compare the gene activities in different types of cells or in different conditions and study how the patterns of gene expression change over time or in response to different stimuli. We can use such information to understand biological activities, normal functions of the cells, effects of pathogens or therapy, or response to any factors. In short, examining mRNA of cells under a specific condition provides a snapshot or a checkpoint of the cellular activities at that moment. Therefore, it is mostly studied to investigate how a certain disease like cancer may affect the gene expression or to study the cellular response to a treatment or a condition.

A gene is a DNA sequence that consists of several functional components; each plays a different role in the process of gene transcription into RNA. In general, the functional gene regions can be divided into two main units: the promoter region and the coding region. The promoter region controls the transcription of the mRNA, whereas coding

region consists of codons that are translated into amino acids. The major difference between prokaryotic and eukaryotic genes is that the coding regions of a eukaryotic gene (also known as exons) are found between the non-coding regions (known as introns) that make gene expression in eukaryotes more complex. In eukaryotes, once mRNA is transcribed, the introns are removed by splicing and the final mRNA transcript will form an open reading frame (ORF) that can be translated into a polypeptide. A single eukaryotic gene may code for several genes by the so-called alternative splicing producing multiple proteins called isoforms. In eukaryotic mRNA, introns are spliced out at specific splicing sites found at the 5' and 3' ends of introns. Most commonly, the intron sequence that is spliced out begins with the dinucleotide GT (donor splice site) at its 5' end and ends with AG (acceptor splice site) at its 3' end [1]. Very rare splice site may be found at AT of 3'-end and AC at the 3'-end of an intron and also at GC (5'-end) and AG (3' end) of an intron [2]. On the other hand, a prokaryotic gene consists of a continuous sequence of codons that makes gene expression in prokaryotes simpler. A typical functional promoter region of a eukaryotic gene consists of the transcription start site (TSS, where Polymerase II binds), a core promoter called TATA box (around 40 nucleobases upstream from the transcription start site), upstream promoters (different in sequence from a gene to another), and the enhancer (may be found thousands of nucleobases upstream or downstream). TATA box is consistent among eukaryotic species; it is the site where transcription factors (TATA box-binding proteins or TBP) bind to form a complex that is necessary for transcription to take place. The upstream promoters can bind to different types of proteins to activate the gene. The enhancer binds to special types of proteins to bend the DNA sequence so the enhancer that can bind to the protein complex on the promoter region.

The prokaryotic genes are organized into blocks called operons. Genes in a single operon are regulated and expressed together. An operon contains all genes that encode proteins which carry out together a specific function. For instance, the bacterial genes required for lactose as energy source are found next to each other in the lactose operon (lac operon). The gene structure and regulation of prokaryotic genes is different from that of eukaryotic ones. There is no intron in prokaryotic genes and an operon, which contains several genes, has a single promoter controlling the gene expression of all genes in that operon. There are three types of proteins (repressors, activators, and inducers) that regulate expression of genes in an operon.

In general, the normally functioning cells of a living organism must be able to control gene transcription by turning on and off genes based on the need of the cells for protein synthesis and other functions. The process of turning on a gene to be transcribed to synthesize functional gene products is known as the gene expression. The activity of a gene is measured by the amount of its transcript (mRNA). Laboratory techniques like northern blot, serial analysis of gene expression (SAGE), quantitative PCR (qPCR), and DNA microarray have been used to study gene expression. However, these techniques can be replaced these days by the high-throughput RNA sequencing, which provide more information than any of the other techniques.

5.2 RNA-SEQ APPLICATIONS

The RNA-Seq emerged as an alternative to microarrays for gene expression research. It uses the high-throughput sequencing of RNA to examine the quantity of RNA in biological samples. The primary application of the RNA-Seq is the gene profiling, which is the determination of the patterns of genes expressed at the level of transcription under specific condition or in a specific type of cells to give an overall picture of biological activities. Simply, it focuses on the transcriptome to tell us which of the genes in the genome of an organism are turned on or off and to what extent. Compared to microarrays, RNA-Seq provides an unbiased information about transcripts and it can detect a larger number of differentially expressed genes. In addition to gene expression profiling, RNA-Seq is also used in a variety of applications, including complex differential gene expression, single-cell RNA-Seq, small RNA profiling, variants identification and allele-specific expression, detection of alternative splicing patterns, system biology, and fusion gene detection.

The differential gene expression is the most important application of RNA-Seq, and it is the application on which we will focus in detail in this chapter. In differential expression analysis, we will compare transcriptomes across different conditions of interest. These conditions can be different samples, development stages, treatments, disease conditions, etc. The sequencing reads produced from the samples are counted to measure the gene expression levels. The read counts then are modeled for the statistical significance that helps us to conclude whether the difference between samples is significant. The differential analysis can also be followed by annotation using functional annotation database such as Gene Ontology (GO) and Kyoto Encyclopedia of Genes and Genomes (KEGG) pathways.

RNA-Seq is also used for the single-cell RNA analysis in which the RNA is extracted from a single cell or a single type of cells. Single-cell RNA sequencing provides transcriptional profiling of individual cells that enable researchers to study the genes expressed in the single-cell level and how expression level varies across thousands of cells within a heterogeneous sample. Most of living cells cannot be cultivated in vitro. Therefore, the single-cell RNA-seq may lead to the discovery of novel species, pathway, or regulatory processes of biotechnological or medical relevance. The workflow of single-cell RNA-Seq generally involves single-cell isolation, cDNA library preparation, RNA-Seq, and RNA-Seq data analysis [3].

The RNA-Seq is also used to study the small non-coding RNA (sRNA-Seq), which is a type of non-coding RNA that includes microRNA, small interfering RNA (siRNA), and P-element-induced wimpy testis (Piwi)-interacting RNA, small nucleolar RNA (snoRNA), and small nuclear RNA (snRNA). The small RNA-Seq data analysis is known to be challenging due to the short length of the sequence and non-unique genomic origin.

The RNA-Seq can also be used for variant detection, allele-specific expression, and expression quantitative trait loci (eQTL). The polymorphisms in coding region (exons) are better to be detected using transcriptomic sequencing. However, the variant calling pipeline must be applied. But the basic RNA-Seq pipeline can be used to quantify the allele-specific expression (ASE) that is affected by single-nucleotide polymorphisms (SNPs). The ASE analysis will show which one of two alleles is highly transcribed into mRNA and

which one is lowly transcribed or even not transcribed at all. The RNA-Seq count data is used as an alternative to microarray data in eQTL analysis. QTL analysis is a statistical method that links phenotypic data (trait measurements) and genotypic data (markers usually SNPs) in an attempt to explain the genetic basis of variation in complex traits. On the other hand, eQTL analysis links markers (genotype) with gene expression levels measured in a large number of individuals and the data is modeled using generalized linear models.

RNA-Seq is a powerful tool for detecting alternative splice patterns, which are important to understand development of human diseases. Paired-end sequencing enables sequence information from both ends and help in detecting splicing patterns without requirement for previous knowledge of transcript annotations. The single-molecule, real-time (SMRT) sequencing is the core technology powering long-read sequencing that allows examination of splicing patterns and transcript connectivity in a genome-scale manner by generating full-length transcript sequences.

RNA-Seq is also used for fusion gene detection. A fusion gene is a gene made by joining two different genes. It is usually created when a gene from one chromosome moves to another chromosome. The fusion gene is transcribed into mRNA that will be translated into fusion protein. The fusion proteins implicate usually in some types of cancer including leukemia; soft tissue sarcoma; cancers of the prostate, breast, lung, bladder, colon, and rectum; and CNS tumors. Paired-end RNA-Seq data are usually used for fusion gene detection [4].

Other kinds of RNA-Seq applications include integration of RNA-Seq data analysis with other technologies.

The library preparation of the mRNA is similar to that of DNA. However, mRNA must be separated from other types of RNA by enrichment technique which uses either PCR amplification or the depletion of the other types of RNA. The RNA must be converted into complementary DNA (cDNA) by reverse transcription before library preparation. As DNA library, the cDNA library preparation involves fragmentation and adaptor ligation to each end of the fragments. The cDNA fragments then are sequenced with the sequencing machine and the sequencing can either be single end (forward strand only) or paired end (forward and reverse strands). The sequencing generates sequence data in a form of reads in FASTQ files. Those reads are the sequenced fragments of the expressed genes in the sample.

5.3 RNA-SEQ DATA ANALYSIS WORKFLOW

The first steps of the RNA-Seq are the same as in other sequencing applications. The sequencing raw data (usually in FASTQ files) must pass through the quality control steps that were discussed in detail in Chapter 1. In general, the steps of the workflow include quality control, read alignment, read quantification, differential expression, annotation, and interpretation (Figure 5.1).

5.3.1 Acquiring RNA-Seq Data

The RNA-Seq raw data are sequence reads produced by a sequencing instrument. RNA-Seq sequence raw data for some projects are available in public databases and can be

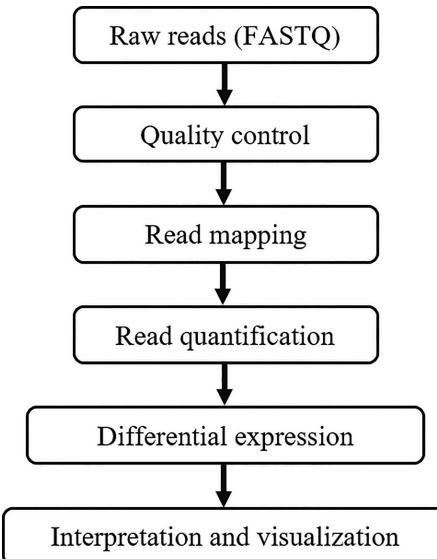


FIGURE 5.1 RNA-seq data analysis workflow.

downloaded and used for research purposes or for learning. Most RNA-Seq raw sequence data are in FASTQ file format. When analyzing the RNA-Seq data, we must pay attention to the design of the study. For instance, if the purpose is the differential gene expression, there must be control raw data that we can use for comparison. The control raw data is determined by the research goal. In conditions like cancers, researchers may use sequencing raw data of healthy tissue as control against the raw data of the affected tissue and both from the same individual. However, researchers may also intend to compare gene expression across individuals or samples. Most researchers include replicate samples in the design of their study, and thus, there will be multiple raw data for a single sample. Replicate samples will reduce errors generated by the laboratory technique used and also the possible errors generated during the sequencing steps.

For practicing, we will use RNA-Seq raw data of a breast cancer study for differential gene expression in tumor cells. The data is in six FASTQ files (three replicates for tumor and three replicates for normal) containing paired-end reads of the size 151 bases. For the sake of simplicity, the files include only the RNA-seq reads of chromosome 22. The data was adapted to be as simple as possible, so its processing does not take too much time. To keep the files organized, create a main directory “rnaseq” to be as the project directory and create inside it the subdirectory “fastq”, and then, inside this subdirectory, download the raw data from “<https://github.com/hamiddi/ngs>”. To avoid repetition, assume that the raw data files have been cleaned from adaptors, duplicates, and the low-quality reads.

5.3.2 Read Mapping

The read mapping follows reprocessing and cleaning of the raw data. The accuracy of analyses depends heavily on the read mapping. The mapping, as discussed in Chapter 2, is the

process of mapping reads to a reference genome producing a SAM/BAM file that contains the mapping information. Refer to Chapter 2 for the read mapping and the content of SAM/BAM files. When dealing with RNA-Seq data, we can either align the reads to a reference genome or a reference transcriptome. When we align RNA-Seq to a eukaryotic reference genome, we must use an aligning program like STAR that is able to detect the splice junctions. The reads in this case will map to the exons leaving introns and other non-coding regions of the genome uncovered. On the other hand, when aligning RNA-Seq reads to a reference transcriptome, the aligned reads may cover the entire sequence. This strategy is preferable when reads are very short (less than 50 bases). The downside of aligning reads to a transcriptome is that we may miss some novel genes since the transcriptome is made up of only known transcripts. As discussed in Chapter 2, there are several aligners; however, for RNA-Seq, we prefer to use a splice-aware aligner that is able to introduce long gaps to span introns when aligning reads to a reference genome. The commonly used aligners for RNA-Seq data include STAR [5], segemehl [6], GEM [7], BWA [8], BWA-MEM [8], and BBMap [9].

Before deciding on which of the aligners to use with RNA-Seq reads, make sure that the aligner is splicing-aware and able to distinguish between reads aligned across exon–intron boundaries and reads with short insertions [10]. The splicing-aware aligners include STAR [5], GSNAp [11], MapSplice [12], RUM [13], and HISAT2 [14]. Each of these aligners has different advantages and disadvantages in terms of memory efficiency, performance, and speed. Refer to the user guide of any of these aligners to learn more about them. We will use STAR (Spliced Transcripts Alignment to a Reference) as an example aligner for aligning RNA-Seq data. Several studies found that STAR is one of the most accurate aligners of RNA-Seq reads [15]. However, STAR requires a large memory for indexing and mapping. The reference sequence must be indexed by STAR before alignment. STAR begins mapping process by aligning the longest reads that exactly match a single or multiple location on the reference sequence. For partially aligned reads, STAR will attempt to align the unmapped region to a different region. Those parts of the reads which align to different locations of the reference sequence are called seeds. If STAR does not find an exact match to a read on the reference sequence, the read will be extended by inserting gaps. If the extension does not give a good alignment, it will be removed. In the second step of the STAR alignment process, multiple seeds will be clustered based on proximity to a set of anchor seeds. The clustered seeds are stitched together based on the best alignment score [5].

When reads are mapped to a reference sequence, the percentage of mapped reads reflects the quality of the alignment. Low percentage indicates contamination of the DNA. Read coverage and depth on exons are other factors that determine alignment quality.

Above, we have downloaded the FASTQ files in the directory “fastq”. We can map reads in the FASTQ files to a reference genome using STAR program. STAR is a short read aligner designed to align RNA-Seq reads to a reference sequence (genome or transcriptome). For aligning reads in the FASTQ files using STAR, we need to download a reference sequence together with its annotation file in GTF format. Since our example FASTQ files are from human samples, we need to download the latest human genome and its annotation file.

To keep the files organized, we will create two subdirectories in our project directory: “refgenome” where we will store the reference genome and “gtf” where we will save the GTF annotation file.

The sequences of reference genomes and annotation are available in many sequence databases such as Ensembl, UCSC, and NCBI genome database. iGenomes built by Illumina has facilitated the process of downloading the reference data for the frequently analyzed organisms. Genome builds in FASTA and their annotation in GTF/GFF files from the above major databases are available for download. The iGenomes website that includes the download links is available at “<https://support.illumina.com/sequencing/sequencing-software/igenome.html>”. Reference data can also be downloaded from “<https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/>”. For aligning with STAR, we will download the UCSC human reference genome sequence in FASTA and gene annotation in GTF file because the chromosomes are indicated by names rather than accession numbers. While you are in the main directory “rnaseq”, run the following bash script to create the subdirectories and to download the human reference genome and its gene annotation:

```
mkdir refgenome
wget \
-O "refgenome/hg38.fa.gz" \
"https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.
fa.gz"
gzip -d refgenome/hg38.fa.gz
mkdir gtf
wget \
-O "gtf/hg38.ncbiRefSeq.gtf.gz" \
"https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/genes/
hg38.ncbiRefSeq.gtf.gz"
gzip -d gtf/hg38.ncbiRefSeq.gtf.gz
```

Mapping reads to a reference genome using STAR is a two-step process: creating a reference sequence index and then mapping reads to the reference sequence.

The following command creates the STAR index for the reference genome sequence. The “--runThreadN” specifies the number of processors to use, “--genomeDir” specifies the directory where the index files will be saved, “--genomeFastaFiles” and “--sjdbGTFfile” specify the directories of the reference genome file and annotation file, respectively, and “--sjdbOverhang” specifies the read length -1 (read length minus one).

```
mkdir indexes
STAR --runThreadN 4 \
--runMode genomeGenerate \
--genomeDir indexes \
--genomeFastaFiles refgenome/hg38.fa \
--sjdbGTFfile gtf/hg38.ncbiRefSeq.gtf \
--sjdbOverhang 150
```

For indexing the human genome, STAR may require more than 32GB of RAM and it may take a long time depending on the computer RAM and the number of processors used.

Once indexing has been created, you can align the reads in FASTQ files to the reference genome sequence. Since the practice sequence data is paired end, there are two FASTQ files (r1 and r2) for each sample. STAR requires these two files as inputs in addition to the reference genome index we created above. Instead of running STAR for each sample, we can use bash script as follows to align the reads of the sample files in “fastq” directory:

```
mkdir starout
mkdir bam
cd fastq
for i in $(ls *.gz | rev | cut -c 13- | rev | uniq);
do
    STAR --runThreadN 4 \
        --runMode alignReads \
        --outSAMtype BAM SortedByCoordinate \
        --readFilesCommand zcat \
        --genomeDir ../indexes \
        --outFileNamePrefix ../starout/${i} \
        --readFilesIn ${i}_r1.fastq.gz ${i}_r2.fastq.gz
done
for f in $(ls *.bam | rev | cut -c 30- | rev);
do
    mv ${f}Aligned.sortedByCoord.out.bam ../bam/${f}.bam
done
cd ..
```

In the above, first we created a directory “starout” for STAR output and “bam” for the BAM files. The Linux bash command “\$(ls *.gz | rev | cut -c 13- | rev | uniq)” lists the names of the FASTQ files found in the directory and cut the common name (ID) for the files of each sample. The “for loop” will provide that common name as a variable “\${i}” to assemble the FASTQ file names and the output file name prefix to the STAR command each time until all FASTQ files are processed. There will be output files including alignment log files and a BAM file for each sample (6 BAM files for all example data). The second “for loop” renamed the long BAM file names and moved the files to the “bam” subdirectory.

The bash scripting comes in handy whenever there is a task that needs to be repeated multiple times.

The STAR output log files provide important statistics about the read alignment. You can refer to STAR manual to read about the output log files.

Before moving to the next step, you need to index the BAM files using “samtools index” command as follows:

```
#index bam
cd bam
for i in $(ls *.bam);
```

```

do
    samtools index ${i}
done
cd ..

```

5.3.3 Alignment Quality Assessment

After aligning the reads to a reference genome and obtaining the alignment information in a BAM file, we may need to check the alignment quality before the next step in the data analysis. The aligned reads may contain duplicate or overrepresented reads produced by the sequencing process. STAR generates log files that contain some important statistics. In our example, the log files of each replicate have been saved in “starout” directory. Open these files and study the statistics included in each file. Figure 5.2 shows alignment statistics in a STAR log file “*final.out”. The log file gives an idea about the mapped reads and splice sites as well as other important statistics.

Mapping speed, Million of reads per hour	9.88
Number of input reads	331956
Average input read length	302
UNIQUE READS:	
Uniquely mapped reads number	275012
Uniquely mapped reads %	82.85%
Average mapped length	294.34
Number of splices: Total	302027
Number of splices: Annotated (sjdb)	298295
Number of splices: GT/AG	301028
Number of splices: GC/AG	697
Number of splices: AT/AC	61
Number of splices: Non-canonical	241
Mismatch rate per base, %	0.43%
Deletion rate per base	0.01%
Deletion average length	1.62
Insertion rate per base	0.00%
Insertion average length	1.44
MULTI-MAPPING READS:	
Number of reads mapped to multiple loci	13283
% of reads mapped to multiple loci	4.00%
Number of reads mapped to too many loci	161
% of reads mapped to too many loci	0.05%
UNMAPPED READS:	
Number of reads unmapped: too many mismatches	0
% of reads unmapped: too many mismatches	0.00%
Number of reads unmapped: too short	43458
% of reads unmapped: too short	13.09%
Number of reads unmapped: other	42
% of reads unmapped: other	0.01%

FIGURE 5.2 STAR alignment statistics.

As shown in Figure 5.2, the alignment statistics of one of the BAM files show the total number of reads, the average reads length, the number and percentage of uniquely mapped reads, splice statistics, statistics of the reads mapped to multiple genes, statistics of the unmapped reads, and chimeric reads. Pay attention to the reads mapped to multiple loci and chimeric; when their number is large, that indicates low-quality alignment. Remember that this BAM file includes the alignments of chromosome 22 only. The number of reads will be huge if the BAM file contains the alignments of all chromosomes.

In addition to the statistics in the STAR log files, there are a variety of programs for assessing alignments in BAM files. Examples of those programs include Qualimap [16], RNA-seQC [17], and RSeQC [18]. Those programs compute metrics for RNA-Seq data, including per-transcript coverage, junction sequence distribution, genomic localization of reads, 5'-3' bias, and consistency of the library protocol. As an example, you can download and use Qualimap to obtain an overall view about the alignment quality on an HTML format. You can download Qualimap from “<http://qualimap.conesalab.org/>” and unzip it in your project directory. Run Qualimap for each sample and study the reports carefully. The following script is an example of how to use it:

```
mkdir qc
qualimap_v2.2.1/qualimap rnaseq \
    -outdir qc \
    -a proportional \
    -bam bam/norm_rep1.bam \
    -p strand-specific-reverse \
    -gtf gtf/hg38.ncbiRefSeq.gtf \
    --java-mem-size=8G
```

The above script creates the directory “qc” where the Qualimap output files will be saved. The program takes a BAM file and the reference annotation file as inputs and generates an HTML report that includes summary statistics about read alignments, reads genomic origin, transcript coverage profile, splice junction analysis, and figures about read genomic origins, coverage profile along genes, coverage histogram, and junction analysis. As a biologist, you may need to study these metrics to have a general idea about the sample alignment before proceeding.

5.3.4 Quantification

Gene profiling or studying gene expression is centered in the quantification of aligned reads per gene or locus. Quantification of reads begins by counting the number of reads aligned to each gene annotated on the sequence of the reference sequence. Given a BAM file with aligned RNA-Seq reads and a list of genomic features in an annotation file (GTF format), the task of the read counting program is to count the number of reads mapping to each feature. In general, a feature, in this case, is a gene which represents a transcript or unions of exons of a gene for eukaryotic organisms. Some programs can also consider exons as features. This is especially useful for checking alternative splicing in the eukaryotic genes. A read in the BAM file may map to a single feature (unique) or may map or

overlap with multiple features (non-unique). A read that maps or overlaps with multiple features is considered as ambiguous and will not be counted. Only reads mapping unambiguously to a single feature are counted.

For RNA-Seq read counts, we can use a variety of programs but the most used free open-source programs include HTSeq-count [19] (Python-based program) and FeatureCounts [20], which is used as a Linux command-line program or as a function in the R package Rsubread. Both these programs require BAM files and GTF file (reference annotation file) as inputs.

In the following, we will use HTSeq-count to count RNA-seq reads aligned to the genes in chromosome 22. Install HTSeq-count by following the installation instructions available at “<https://htseq.readthedocs.io/en/master/install.html>”. The following HTSeq-count command counts aligned reads in all BAM files stored in the “bam” subdirectory and saves the output in “features/htcount.txt”. The “-m union” option specifies the union mode to handle reads overlapping more than one feature. We used “--additional-attr=transcript_id” to add transcript accession numbers to the output.

```
mkdir features
htseq-count \
-m union \
-f bam \
--additional-attr=transcript_id \
-s yes bam/*.bam \
-gtf/hg38.ncbiRefSeq.gtf \
> features/htcount.txt
sed '/^__/ d' < htcount.txt > htcount2.txt
```

The “sed” command has been used to remove the last rows that begin with “__” and to save that change in a new file “htcount2.txt”, which we will use in the next step of the analysis.

ACKR2	NM_001296.5	0	0	0	0	0
ACKR3	XM_005246098.3	0	0	0	0	0
ACKR4	NM_016557.4	0	0	0	0	0
ACKR4P1	ACKR4P1	0	0	0	0	0
ACLY	XM_017024688.1	0	0	0	0	0
ACMSD	XM_005263586.4	0	0	0	0	0
ACNATP	ACNATP_2	0	0	0	0	0
AC01	NM_001278352.2	0	0	0	0	0
AC02	XM_024452250.1	76	106	82	269	291
ACOD1	NM_001258406.2	0	0	0	0	0
ACOT1	XM_017021591.1	0	0	0	0	0
ACOT11	NM_015547.4	0	0	0	0	0
ACOT12	XM_017009047.1	0	0	0	0	0
ACOT13	NM_001160094.2	0	0	0	0	0
ACOT2	NR_046028.1	0	0	0	0	0
ACOT4	XR_001750151.2	0	0	0	0	0
ACOT6	NM_001365789.1	0	0	0	0	0

FIGURE 5.3 HTSeq-count feature count.

The HTSeq-count output file contains the feature count for each sample as shown in Figure 5.3. The feature count file includes tab-delimited columns for gene symbols, transcript IDs, and a count column for each sample. We can notice that some genes have zero reads aligned to them. Later, we will filter out the genes that have no aligned reads and the one with low coverage.

5.3.5 Normalization

In general, when we analyze gene expression data, we may need to normalize it to avoid some biases that may arise due to the gene lengths, GC contents, and library sizes (the total number of reads aligned to all genes in a sample) [21]. The normalization of count data is important for comparing between expression of genes within the samples and between different samples. The normalized gene length fixes the bias that may affect within-sample gene expression comparison. It is known that a longer gene would have a higher chance to be sequenced than a shorter gene. Consequently, a longer gene would have a higher number of aligned reads than a shorter one at the same gene expression level in the same sample. The GC content also affects within-sample comparison of gene expressions. The GC-rich and GC-poor fragments tend to be under-represented in RNA-Seq sequencing, and hence, the gene with the GC content closest to 40% would have higher chance to be sequenced [22]. The library size affects the comparison between the expressions of the same gene in different samples (between-sample effect).

There are several normalization methods for adjusting the biases resulted from the above-mentioned possible causes. Choosing the right normalization method depends on whether the comparison is within-sample or between-samples. In the following, we will discuss some of these normalization methods used by gene expression analysis program like EdgeR and DESeq2 [23].

5.3.5.1 RPKM and FPKM

RPKM [24] (the reads per kilobase of transcript per million reads mapped) is a normalized unit for the counts of reads aligned to genes (normalized gene expression unit). It scales the count data by gene length to adjust for the sequencing bias arising from the differences in gene lengths. The RPKM is used for within-sample gene expression comparison (i.e., comparison between genes in the same sample).

Assume that N reads are aligned to the reference sequence and only k reads are aligned to the gene i of length l_i bp, the PRKM of the gene i is calculated as

$$\text{RPKM}_i = \frac{k_i}{\left(\frac{l_i}{10^3}\right)\left(\frac{N}{10^6}\right)} \quad (5.1)$$

In the denominator of Formula 5.1, the length of gene (l) (in base) is divided by 1000 to be in kilobase and the total number of reads aligned to the reference sequence is divided by 1000,000 (million). When the number of reads aligned to a gene is divided by the

denominator, we will obtain the reads per kilobase per million or RPKM. The RPKM is for single-end reads. However, for paired-end reads, both forward and reverse reads are aligned, and thus, “fragment” is used instead of “read” and the normalized unit of gene expression in this case is FPKM (fragment per kilobase per million).

5.3.5.2 Transcripts per Million

The transcripts per million (TPM) [25] is proposed as an alternative to RPKM and FPKM to adjust for the bias of gene length and to be used for within-sample differential gene expression. The TPM represents the abundance of reads aligned to gene i in relation to the abundance of the reads aligned to other genes in the same sample. To normalize the RNA reads counts, first, for any gene, divide the number of reads aligned to it by its length, forming the count per gene length in base. Then, divide the count per length in base by the sum of all counts (per length in base of every gene) and multiply by 1000,000 forming the transcript per million or TPM.

$$\text{TPM}_i = \frac{k_i}{l_i} \times \frac{1}{\left(\sum_j \frac{k_j}{l_j} \right)} \times 10^6 \quad (5.2)$$

5.3.5.3 Counts per Million Mapped Reads

The counts per million (CPM) mapped reads normalize the number of reads that map to a particular gene after correcting for sequencing depth and transcriptome composition bias [26]. CPM is used for between-sample differential analysis to compare between the gene expressions of the same gene in different samples. It is not suitable for within-sample gene expression comparison because it does not adjust for the gene length. The CPM of a gene is defined as the number of reads mapped to a gene divided by the total number of mapped read (or library size) multiplied by 1000,000.

$$\text{RPM}_i = \frac{k_i}{N} \times 10^6 \quad (5.3)$$

5.3.5.4 Trimmed Mean of M-values

The Trimmed Mean of M-values (TMM) [27] is used by edgeR for between-sample differential gene expression. It uses the relative gene expression of two samples: one is the sample of interest (treated), and the other is the reference that we wish to use as a baseline for comparison. The gene-wise log-fold change of gene g M_g is given as:

$$M_g = \log_2 \left(\frac{\frac{Y_{gi}}{K_j}}{\frac{Y_{gr}}{K_r}} \right) \quad (5.4)$$

where Y_{gi} is the observed read count of the gene g of interest in the sample j , K_j is the size of library of sample j (total number of aligned reads), and Y_{gr} is the observed reads count of the same gene g in the reference sample r of library size K_r .

Then, the value of the gene expression fold change, M_g , is trimmed by 30% followed by taking the weighted average for the trimmed M_g using inverse of the variances of read counts of genes (V_{gi}) as weight since the log-fold changes from gene with larger read counts will have lower variance on the logarithm scale. The TMM adjustment, f_j , is given as

$$f_j = \frac{\sum_i^n V_{gi} M_{gi}}{\sum_i^n V_{gi}} \quad (5.5)$$

The adjustment f_j is an estimate for relative RNA production of two samples. The TMM normalization factor for the sample j with m genes is given by

$$N_j = \sqrt{f_j} \sum_{g=1}^m Y_{gi} \quad (5.6)$$

TMM does not correct the observed read counts for the gene length, and hence, it is not suitable for comparison between the gene expressions in the same sample.

5.3.5.5 Relative Expression

For a given sample j , the relative expression (RE) scaling factors are calculated as the median of the ratios of observed counts to the geometric mean across all samples (pseudo-reference sample, r) [28]. The scaling factors are calculated as follows:

$$N_j = \frac{\text{median}_g \frac{y_{gi}}{\left(\prod_{r=1}^n Y_{gr}\right)^{1/n}}}{g} \quad (5.7)$$

5.3.5.6 Upper Quartile

The upper quartile (UQ) normalization factor is computed as the sample upper quartile (75th percentile) of gene counts for the genes with no zero counts for all samples [29].

5.3.6 Differential Expression Analysis

Most differential expression programs have functions that normalize gene expression count data as part of the analysis. The design of the study is crucial in the differential expression analysis as we discussed in the introduction of this chapter. The conditions that group the samples and sample replicates must be determined as metadata before the analysis. A simple gene expression study is made up of two groups (e.g., treated and control

or healthy and diseased) but it can also be a complex study that includes more than a single factor (factorial design). Once the study design has been determined as metadata, inferential statistics is used to identify which gene has statistically significant change in expression compared to the same gene in a reference sample. The fundamental step in the differential expression analysis is to model the association between gene counts (Y) and the covariates (conditions) of interest (X). The number of replicates is crucial for the statistical differential analysis. Most of the time, the number of replicates in an RNA-Seq study is small. Instead of non-parametric statistical analysis, most programs for RNA-Seq data analysis use generalized linear models (GLMs) by assuming that the count data follows a certain statistical distribution. That approach also assumes that each RNA-Seq read is sampled independently from a population of reads and the read is either aligned to the gene g or not. When the read is aligned to the gene g , we call this a success and otherwise is a failure. The process of random trials with two possible outcomes (success or failure) is called Bernoulli's process. Thus, according to the probability theory, the number of reads (successes), Y_g , for a given gene g from sample j follows a binomial distribution.

$$Y_{gj} \sim \text{Binomial}(n_j, \pi_{gj}) \quad (5.8)$$

Assume Y_{gj} is the number of reads sequenced from sample j , n_j represents the number of independent trials in Bernoulli's process, π_{gj} is the probability of success (a read is aligned to the gene g in sample j), and $1 - \pi_{gj}$ is the probability of failure

Assume also that for the gene g on the sample j and that gene has the length l_g and read count Y_{gj} . All possible positions in g that can produce a read can be described as $Y_{gj} l_g$ [30]. Thus, probability of success, π_{gj} , is given as

$$\pi_{gj} = \frac{Y_{gj} l_g}{\sum_{g=1}^G Y_{gj} l_g} \quad (5.9)$$

where G is the number of genes in the sample.

According to the binomial distribution, the mean of the read counts is given as

$$\mu_{gj} = n_j \times \pi_{gj} \quad (5.10)$$

The probability that the number of reads ($X_{gj} = x_{gj}$) for a given gene is given by

$$P(Y_{gj} = y_{gj}) = \binom{n_j}{y_{gj}} \pi_{gj}^{y_{gj}} (1 - \pi_{gj})^{n_j - y_{gj}} \quad (5.11)$$

However, since in RNA-Seq count data, a very large number of reads are represented and the probability of aligning a read to a gene is very small, the Poisson distribution is more appropriate than the binomial distribution if the mean of read counts of a gene is equal to the variance as the Poisson distribution assumes.

$$Y_g \sim \text{Poi}(\lambda_g) \quad (5.12)$$

where λ_g is the Poisson parameter which represents the rate of change in the count of the gene g in a sample.

The Poisson distribution assumes that the rate of change is equal to the mean, λ_g , which is also equal to the variance.

$$\mu_g = \lambda_g = \text{var}(Y_g) \quad (5.13)$$

The probability that $Y_g = y_g$

$$p(Y_g = y_g; \lambda_g) = \frac{\lambda_g^{y_g} \times e^{-\lambda_g}}{y_g!} \quad (5.14)$$

To model the RNA-Seq count data with the Poisson distribution it requires that the mean is equal to the variance. A key challenge is the small number of replicates in typical RNA-Seq experiments (two or three replicates per condition). Therefore, inferential methods that deal with each gene separately may suffer, in this case, from lack of power, due to the high uncertainty of within-group variance estimates. This challenge can be overcome either by grouping the count data into groups and then calculating the variance and the mean in each group or by pooling information across genes by assuming the similarity of the variances of different genes measured in the same experiment. In general, RNA-Seq count data suffers from over-dispersion, where variance is greater than the mean. There are a variety of software that use different technique for modeling the RNA-Seq count data, but most of them use quasi-Poisson, negative binomial, or quasi-negative binomial distribution, which deal with over-dispersed data.

The quasi-Poisson is similar to the Poisson distribution, but the variance is linearly correlated to the mean of the counts [31].

$$Y_g \sim q\text{Poi}(\mu_g, \theta_g) \quad (5.15)$$

$$\text{var}(Y_g) = \theta_g \mu_g \quad (5.16)$$

where θ_g is the dispersion parameter and μ_g is the mean count.

In the negative binomial distribution, the variance is the function of the mean as

$$Y_g \sim NB(\mu_g, \alpha_g) \quad (5.17)$$

$$\text{var}(Y_g) = \mu_g + \alpha \mu_g^P \quad (5.18)$$

where α is the dispersion parameter and P is an integer but commonly we use $P=2$ (NB2 or quadratic model).

$$\text{var}(Y_g) = (\mu_g + \alpha\mu_g^2) \quad (5.19)$$

In the quasi-negative binomial distribution, the variance is modeled as follows:

$$\text{var}(Y_g) = \sigma_g^2 (\mu_g + \theta\mu_g^2) \quad (5.20)$$

The RNA-Seq study design may include a single or several conditions called factors. A researcher usually may be interested in testing the effect of a condition. For instance, assume that a researcher wants to study breast cancer in women. She conducted an RNA-Seq study on samples from healthy and cancer tissues of five affected women. The analysis programs require a matrix that describes the design called a design matrix. The design matrix defines the model (structure of the relationship between genes and explanatory variables), and it is also used to store values of the explanatory variable [32]. The design matrix will be created from the study metadata as shown in Table 5.1.

The design matrix will include dummy variables setting the level of each factor to either zero or one as we will see soon.

The generalized linear model will fit the data of this study design so that the expression of each gene will be described as a linear combination of the dummy explanatory variables.

$$y = \beta_0 + \beta_1 * \text{Patient} + \beta_2 * \text{Condition} + \varepsilon \quad (5.21)$$

where y is the response variable that represents the gene expression in a specific unit, β_0 is the intercept or the average gene expression when the other parameters are zero, and β_1 and β_2 are the generalized linear regression parameters that represent the effect of each explanatory variable. A log-linear model is used as

$$\log \mu_{gi} = X_i^T \beta_g + \log N_i \quad (5.22)$$

where X_i^T is a vector of covariates (explanatory variables) that specifies the conditions/factors applied to sample i and β_g is a vector of regression coefficients for the gene g .

TABLE 5.1 Sample Information or Metadata for the Design Matrix

SampleID	Condition	Patient
norm_rep1	Norm	Rep1
norm_rep2	Norm	Rep2
norm_rep3	Norm	Rep3
tumo_rep1	Tumo	Rep1
tumo_rep2	Tumo	Rep2
tumo_rep3	Tumo	Rep3

5.3.7 Using EdgeR for Differential Analysis

EdgeR (Empirical Analysis of Digital Gene Expression Data in R) is an R Bioconductor package for differential expression analysis of RNA-Seq data. It performs differential expression of replicated count data using generalized linear model for the over-dispersed count data and the models account for both biological and technical variability. EdgeR uses negative binomial distribution to model the RNA-Seq count data.

Assume that for each sample i , the total number of reads (library size) is N_i , ϕ_g is the dispersion coefficient, and p_{gi} is the relative abundance of gene g in the experimental group i . The mean and variance are estimated as follows:

$$\mu_{gi} = N_i p_{gi} \quad (5.23)$$

$$\text{variance} = \mu_{gi} (1 + \mu_{gi} \phi_g) \quad (5.24)$$

For differential expression analysis using the negative binomial regression, the parameters of interest are the relative abundance of each gene (p_{gi}).

As we have discussed above, the negative binomial distribution changes to the Poisson distribution when the count data is not dispersed ($\phi_g = 0$) or to the quasi-Poisson distribution if the variance is linearly correlated to the mean. EdgeR estimates the dispersion (ϕ_g) as the coefficient of variation (CV) of biological variation between the samples. Dispersion means biological coefficient of variation (BCV) squared that is estimated by dividing Formula (5.24) by μ_{gi}^2 .

$$CV^2 = 1 / \mu_{gi} + \phi_g \quad (5.25)$$

EdgeR calculates the common dispersion for all genes and it can also calculate gene-wise dispersions and then it shrinks them toward a consensus value. Differential expression is then assessed for each gene using an exact test for over-dispersed data [33].

In the following, we will analyze the non-normalized count data obtained by HTSeq-count program in the previous step and saved as “htcount.txt” file in the “features” directory. The analysis will be carried out in R. Therefore, R must be installed on your computer. The instructions of R installation are available at “<https://cran.r-project.org/>”. You can also use R on Anaconda as well. We assume that you have R installed on your computer and it is running. On R, you will also need to install Limma and EdgeR Bioconductor packages by following the installation instructions available at “<https://bioconductor.org/packages/release/bioc/html/edgeR.html>” to install EdgeR and “<https://bioconductor.org/packages/release/bioc/html/limma.html>” to install limma. For the current versions, open R, and on the R shell, run the following:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("edgeR")
BiocManager::install("limma")
```

Once you have R, EdgeR, and limma package installed, you will be ready for the next steps of the differential analysis which can be broken down into the following steps.

5.3.7.1 Data Preparation

For differential analysis, EdgeR requires the count data file and a sample info file. We have already created the count data file in the previous step, but we need to create the sample info file that describes the design of the study. We can create the sample info file manually as shown in Table 5.1. The sample info file is tab-delimited, and the first column contains the unique sample IDs or the BAM file names. Additional columns can contain the conditions or factors depending on the study design. For our example data, we can create the sample info file by executing the following bash script while you are in the main project directory:

```
cd bam
ls *.bam \
| rev \
| cut -c 5- \
| rev > tmp.txt
echo -e "sampleid\tcondition\tpatient" \
> ../features/sampleinfo.txt
awk -F ' '_ '{print $1 "_" $2 "\t" $1 "\t" $2}' \
tmp.txt@ ../features/sampleinfo.txt
rm tmp.txt
cd ../features
```

This script creates the sample info file from the BAM file names and saves it in the “features” subdirectory together with the read count data. For your own data, you may need to modify this script or you can create yours using Linux bash commands or manually.

Then, you need to open R, make the “features” directory as the working directory, and load both limma and edgeR packages.

```
library(limma)
library(edgeR)
```

Load both the count data file and sample info file to the R session as data frame.

```
seqdata <- read.delim("htcount2.txt", stringsAsFactors=FALSE)
sampleinfo <- read.delim("sampleinfo.txt", stringsAsFactors=FALSE)
```

Run the following command to display the first rows of the count data frame:

```
head(seqdata)
```

You will notice that the first two columns are the gene symbol and the transcript IDs. The other six columns contain the read counts. In the next step, we need to separate the count

columns in a different data frame called “countdata” and then we need to add column names and row names to that count data frame. The row names can be the transcript IDs and the column names can be the sample IDs as listed in the sample info file. The sample IDs in the sample info file must be in the same order as the columns of the read counts in the read count file.

```
countdata0 <- seqdata[,-(1:2)]
head(countdata0)
```

The “head” function will display the first rows of the “countdata0” data frame. You can notice that, as shown in Figure 5.4, the data frame is without row names and that the column names do not indicate the sample names. You can also notice that there are numerous rows with all columns being zeros. This is mainly because we have aligned reads for chromosome 22 only.

The second step is to make the gene symbol as the row names and sample IDs as column names of the “countdata0” data frame and to remove rows with zero for all samples (Figure 5.5).

```
rownames(countdata0) <- seqdata[,1]
colnames(countdata0) <- sampleinfo$sampleid
countdata <- countdata0[rowSums(countdata0[])>0,]
head(countdata)
```

After creating a count data frame as in Figure 5.5, the next step is to create a DGEList object to hold the read counts that will be analyzed by EdgeR. The DGEList object is a container for the count data and the associated metadata, including sample names, sample, group, library size, and normalization factors. The DGEList for our example data is created by the following:

```
group = factor(sampleinfo$condition)
y <- DGEList(countdata, group=group)
```

Figure 5.6 shows the created DGEList object. At this time, it holds two slots: counts and samples. The counts slot contains the count data and the samples slot contains the sample

```
> head(countdata0)
  x0 x0.1 x0.2 x0.3 x0.4 x0.5
1  0    0    0    0    0    0
2  0    0    0    0    0    0
3  0    0    0    0    0    0
4  0    0    0    0    0    0
5  0    0    0    0    0    0
6  0    0    0    0    0    0
```

FIGURE 5.4 The count data frame without row name and column names.

```
head(countdata)
      norm_rep1 norm_rep2 norm_rep3 tumo_rep1 tumo_rep2 tumo_rep3
A4GALT      4          1          2          0          0          2
ABHD17AP4    0          0          0          0          1          0
ACO2        76         106         82         269         291         285
ACTBP15      0          0          0          1          0          0
ADA2         1          0          0          1          0          0
ADM2        5          2          0          0          0          0
```

FIGURE 5.5 The data frame after adding row and column names and removing rows with all zeros.

```
> y
An object of class "DGEList"
$counts
      norm_rep1 norm_rep2 norm_rep3 tumo_rep1 tumo_rep2 tumo_rep3
A4GALT      4          1          2          0          0          2
ABHD17AP4    0          0          0          0          1          0
ACO2        76         106         82         269         291         285
ACTBP15      0          0          0          1          0          0
ADA2         1          0          0          1          0          0
627 more rows ...

$samples
  group lib.size norm.factors
norm_rep1  norm    15999          1
norm_rep2  norm    15488          1
norm_rep3  norm    15501          1
tumo_rep1  tumo    15165          1
tumo_rep2  tumo    15297          1
tumo_rep3  tumo    15364          1
```

FIGURE 5.6 The DGEList object of the count data.

info (sample id, group, library size (lib.size), and normalization factors (norm.factor)) with 1 values. This data will be updated soon, and more slots will be added as well.

5.3.7.2 Annotation

The row names of the count data frame are the gene symbols as shown in Figure 5.5. For some of the downstream analysis, we may need the count data to be annotated with the NCBI Entrez IDs and full gene names which are not included in the count dataset at this point. To add these annotations to the DGEList object (y), we need to make the Entrez IDs as the row names instead of the gene symbols. To obtain the Entrez IDs and gene names, we need to install and load the “org.Hs.eg.db” Bioconductor package, which is a genome-wide annotation for human based on mapping using Entrez Gene identifiers [34]. You can install and upload this package by running the following script on R prompt:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("org.Hs.eg.db")
library(org.Hs.eg.db)
```

```
> columns(org.Hs.eg.db)
[1] "ACCCNUM"      "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
[6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
[11] "GENETYPE"     "GO"          "GOALL"        "IPI"          "MAP"
[16] "OMIM"         "ONTOLOGY"    "ONTOLOGYALL" "PATH"         "PFAM"
[21] "PMID"         "PROSITE"     "REFSEQ"       "SYMBOL"      "UCSCKG"
[26] "UNIPROT"
```

FIGURE 5.7 Annotation columns available on “org.Hs.eg.db”.

```
> head(y$genes)
ENTREZID SYMBOL GENENAME
1 53947 A4GALT alpha 1,4-galactosyltransferase (P blood group)
2 729495 ABHD17AP4 ABHD17A pseudogene 4
3 50 ACO2 aconitase 2
4 391334 ACTBP15 ACTB pseudogene 15
5 51816 ADA2 adenosine deaminase 2
6 79924 ADM2 adrenomedullin 2
```

FIGURE 5.8 Adding annotation to the count data.

The available annotation column names are displayed with the following (Figure 5.7):

```
columns(org.Hs.eg.db)
```

Each of these annotation columns has a row value corresponding to the gene annotated in the reference sequence. We can select the annotation columns that we need and add an annotation slot with the selected columns to the DGEList object. The following script creates a vector of the Entrez IDs mapped to the gene symbol on the counts data, makes the Entrez IDs as the row names, selects annotation columns mapped to the count data, adds the annotation as a slot to the DGEList object, and finally removes any row without an Entrez ID:

```
ENTREZID <- mapIds(org.Hs.eg.db, rownames(y) ,
                     keytype="SYMBOL", column="ENTREZID")
rownames(y$counts) <- ENTREZID
ann<-select(org.Hs.eg.db, keys=rownames(y$counts) ,
            columns=c("ENTREZID", "SYMBOL", "GENENAME"))
head(ann)
y$genes <- ann
i <- is.na(y$genes$ENTREZID)
y <- y[!i, ]
```

Figure 5.8 shows the annotation slot “genes” that includes Entrez IDs, gene symbols, and gene names mapping to the count data “counts”.

5.3.7.3 Design Matrix

The design matrix includes dummy variables that define the covariates of the model, depending on the study design to answer specific research questions. We will define the

design matrix using the sample information in the “sampleinfo.txt” that we have created above. In EdgeR, the design matrix can be defined with or without an intercept. The intercept is used when there is a reference for the differential expression analysis. When the design matrix is defined without an intercept, the differential analysis can be performed by using a contrast as we will do. In the following, we define a design matrix without an intercept (Figure 5.9):

```
condition <- factor(sampleinfo$condition)
design <- model.matrix(~ 0 + condition)
design
```

This design matrix defines two dummy variables representing the levels of the condition studied (1 if the condition is correct and zero otherwise). When we fit a negative binomial generalized log-linear model described in Formula 22, two coefficient estimates will be calculated; one for each dummy variable.

5.3.7.4 Filtering Low-Expressed Genes

Some genes may not be expressed or may not have enough reads to contribute to the differential analysis. Therefore, it is good practice to retain only the genes that have sufficient read counts by filtering out the genes with zero or low counts keeping only the ones with at least one count per million (1 cpm) reads in at least two samples. The following script filters out the genes with low abundance and adjust the library size to reflect the new change:

```
keep <- filterByExpr(y, design)
y <- y[keep, , keep.lib.sizes=FALSE]
```

As shown in Figure 5.10, after filtering, the counts slot contains only genes with sufficient abundance and the library size in the samples slot has been adjusted. Notice the difference in the number of genes and library size between Figures 5.10 and 5.6. The new counts slot contains only 133 genes compared to 632 genes before filtering and the library sizes have been adjusted to reflect the new ones.

```
> condition <- factor(sampleinfo$condition)
design <- model.matrix(~ 0 + condition)
design
  conditionnorm conditiontumo
1          1          0
2          1          0
3          1          0
4          0          1
5          0          1
6          0          1
attr(,"assign")
[1] 1 1
attr(,"contrasts")
attr(,"contrasts")$condition
[1] "contr.treatment"
```

FIGURE 5.9 Design matrix without intercept.

```

> keep <- filterByExpr(y, design)
y <- y[keep, , keep.lib.sizes=FALSE]
> y
An object of class "DGEList"
$counts
      norm_rep1 norm_rep2 norm_rep3 tumo_rep1 tumo_rep2 tumo_rep3
50          76        106        82        269        291        285
646023      166        147       140         31         17         35
79087          7          8          3        60        43        65
129138          6          5          2        21        17        14
100874530      78        81        84        48        51        34
127 more rows ...

$samples
      group lib.size norm.factors
norm_rep1  norm     14786        1
norm_rep2  norm     14230        1
norm_rep3  norm     14263        1
tumo_rep1 tumo     14229        1
tumo_rep2 tumo     14404        1
tumo_rep3 tumo     14371        1

$genes
  ENTREZID      SYMBOL          GENENAME
3      50      ACO2      aconitase 2
8    646023  ADORA2A-AS1  ADORA2A antisense RNA 1
11    79087      ALG12  ALG12 alpha-1,6-mannosyltransferase
13   129138    ANKRD54      ankyrin repeat domain 54
17  100874530 APOBEC3B-AS1  APOBEC3B antisense RNA 1
127 more rows ...

```

FIGURE 5.10 DGEList object after filtering out genes with low gene expression.

5.3.7.5 Normalization

After filtering the low-expressed genes, we can normalize the count data. EdgeR uses TMM to compute normalization factor that corrects sample-specific biases. Without normalization, if only few genes have high expression, those genes will account for a substantial proportion of the library size for a specific sample, causing other genes to be under-represented. The normalization factor is multiplied by the library size to yield the effective library size, which is used for normalization. The following function calculates the TMM normalization factor:

```
yNorm <- calcNormFactors(y)
```

Notice that as shown in Figure 5.11, the normalization factor was changed for each sample.

5.3.7.6 Estimating Dispersions

The next step is to use the above normalized count data to estimate the dispersions which will be used to estimate the parameters of the negative binomial model as discussed above. As there are only few replicates or samples, estimation of the gene-wise dispersions based on the count vector of the gene across replicates will not be accurate. EdgeR uses information sharing between genes to estimate dispersion; genes of closely similar abundance will

```

> yNorm <- calcNormFactors(y)
> yNorm
An object of class "DGEList"
$counts
      norm_rep1 norm_rep2 norm_rep3 tumo_rep1 tumo_rep2 tumo_rep3
50          76        106        82       269        291        285
646023      166        147       140        31        17        35
79087         7         8         3       60        43        65
129138         6         5         2       21        17        14
100874530     78        81       84        48        51        34
127 more rows ...

$samples
  group lib.size norm.factors
norm_rep1  norm    14786  0.9042702
norm_rep2  norm    14230  1.0182328
norm_rep3  norm    14263  0.9038986
tumo_rep1 tumo    14229  1.0569558
tumo_rep2 tumo    14404  1.0564503
tumo_rep3 tumo    14371  1.0760416

```

FIGURE 5.11 DGEList object after computing the normalization factor.

```

> yNorm <- estimateDisp(yNorm, design)
names(yNorm)
[1] "counts"           "samples"          "genes"
[4] "design"           "common.dispersion" "trended.dispersion"
[7] "tagwise.dispersion" "AveLogCPM"      "trend.method"
[10] "prior.df"        "prior.n"         "span"

```

FIGURE 5.12 DGEList objects slot including the estimated dispersions.

share a dispersion. Three types of dispersions are calculated: a common estimate across all genes, mean-variance trend dispersion using genes' similar abundance, and gene-wise dispersion (tagwise dispersion).

The “estimateDisp(DGEList, design)” function estimates the common, trended, and tagwise negative binomial dispersions by using weighted likelihood empirical Bayes algorithm [33]. This function requires a DGEList object with normalized counts and a design matrix.

```

yNorm <- estimateDisp(yNorm, design)
names(yNorm)

```

The three dispersions will be estimated and stored in the DGEList object (*y*) as shown in Figure 5.12. You can display the common, trended, and tagwise dispersions by using the following (Figure 5.13):

```

yNorm$common.dispersion
head(yNorm$trended.dispersion)
head(yNorm$tagwise.dispersion)

```

```
> yNorm$common.dispersion
head(yNorm$trended.dispersion)
head(yNorm$tagwise.dispersion)
[1] 0.008864637
[1] 0.007720281 0.010047994 0.015448455 0.035206539 0.012336105 0.035731961
[1] 0.004316713 0.011238688 0.018325542 0.024397190 0.009908097 0.014062947
```

FIGURE 5.13 Printing dispersions.

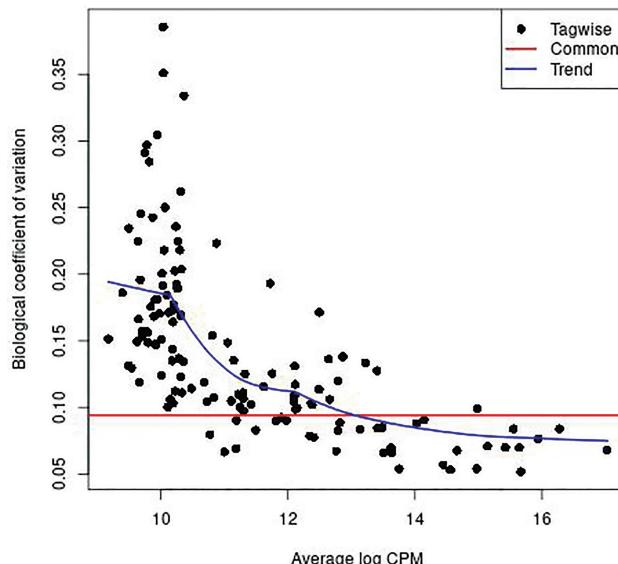


FIGURE 5.14 Biological CV plot.

We can plot the BCV against gene abundance (in log2 counts per million) using “plotBCV(y)” function. As described above, the BCV is the square root of the dispersion parameter in the negative binomial model.

```
jpeg('myBCVPlot.jpg')
plotBCV(yNorm, pch=16, cex=1.2)
dev.off()
```

The above script plots the biological coefficient of variations (based on the three types of dispersions) against the average abundance of each gene. As shown in Figure 5.14, the plot shows the square root estimates of the common, trended, and tagwise negative binomial dispersions.

For RNA-Seq count data, the negative binomial dispersions tend to be higher for genes with very low counts and the dispersion trend tends to decrease smoothly with the abundance (CPM) increase and becomes asymptotic to a constant value for genes with a larger abundance.

5.3.7.7 Exploring the Data

Up to this point, we have prepared the count data for fitting the negative binomial generalized log-linear model. However, before that step, we can explore the data by visualizing the library size and the distribution of cpm (log2 counts per million) of each sample.

```
png(file="libsizeplot.png")
x<-barplot(yNorm$samples$lib.size/1e06,
            names=colnames(yNorm),
            las=2, ann=FALSE,
            cex.names=0.75,
            col="lightskyblue",
            space = .5)
mtext(side = 1, text = "Samples", line = 4)
mtext(side = 2, text = "Library size (millions)", line = 3)
title("Barplot of library sizes")
dev.off()
```

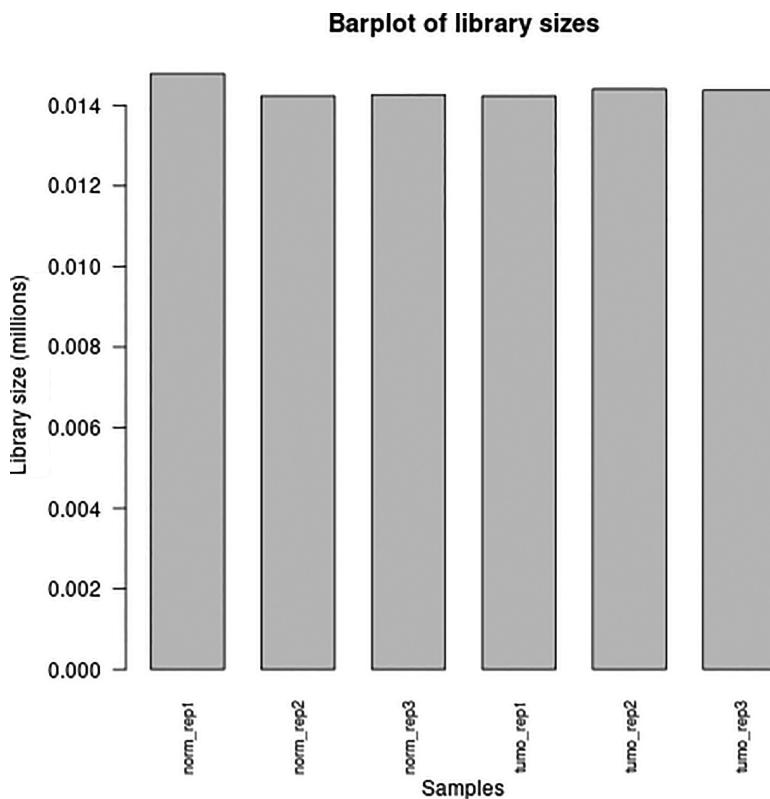


FIGURE 5.15 Library sizes.

As shown in Figure 5.15, the library sizes or the sequencing depths of the six samples are similar. This bar chart gives an idea about the distribution of the library sizes and any potential source of bias from the library sizes.

In the normalization step, we normalized the count data to eliminate composition biases between libraries. We can assess the TMM normalization by the MD plot (mean-difference plot), which displays the library size-adjusted log-fold change (difference) between two libraries against the average log-expression across these libraries (the mean). The points on the MD plot should be centered at a line of zero log-fold change if the biases between libraries were removed successfully by the normalization. The “plotMD(y, column=i)” function creates MD plot by converting the count (y) to log2-CPM values and then creating an artificial array by averaging all samples other than the sample specified (column=i) in the

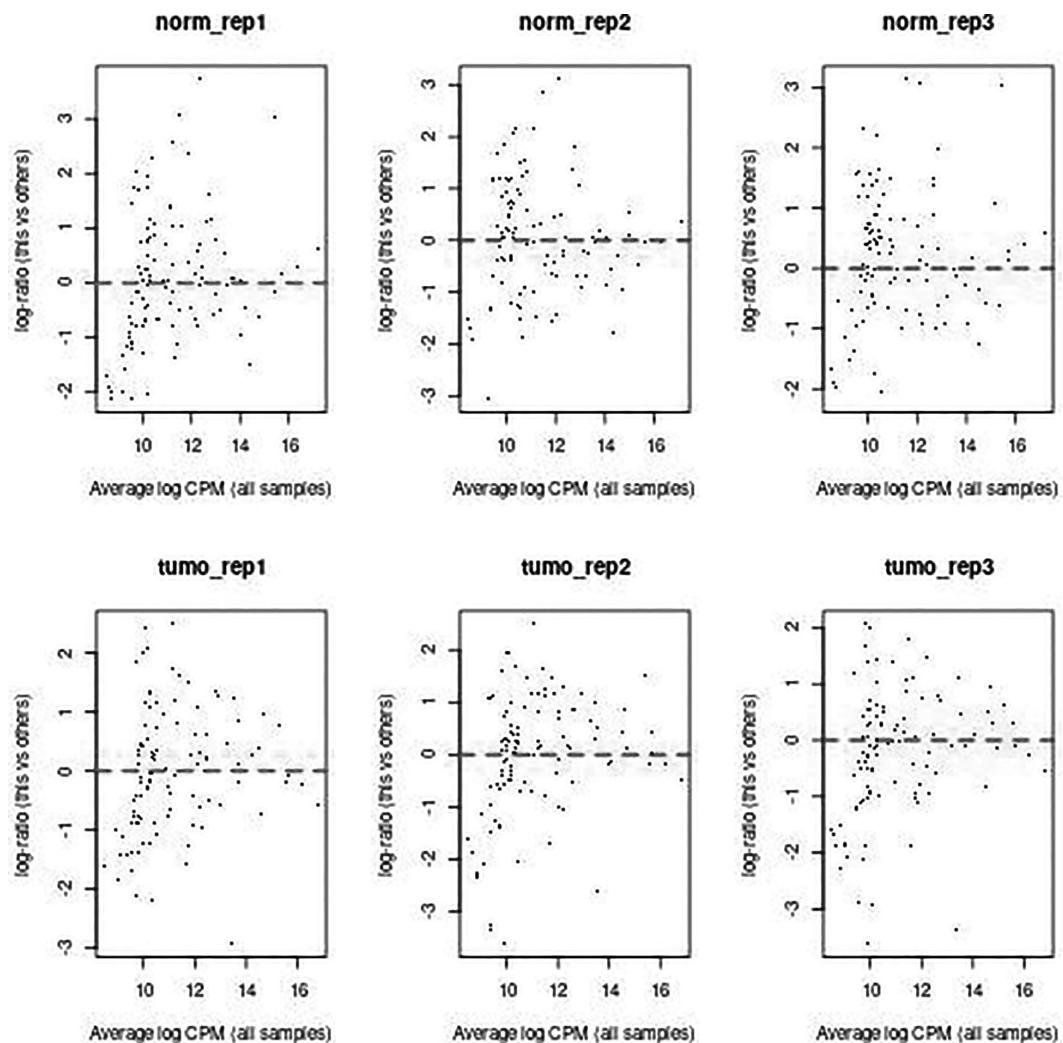


FIGURE 5.16 Mean-difference plots.

function. The function then creates the MD plot from the specified sample and the artificial sample. The following script creates the MD plots for the six samples (Figure 5.16):

```
jpeg('mdplots.jpg')
par(mfrow=c(2,3))
for (i in 1:6) {
  plotMD(yNorm, column=i,
         xlab="Average log CPM (all samples)",
         ylab="log-ratio (this vs others)")
  abline(h=0, col="red", lty=2, lwd=2)
}
dev.off()
```

We can also create boxplots for the unnormalized and normalized log-CPM values to show the expression distributions in each sample using the following script:

```
png(file="logcpmboxplot.png")
par(mfrow=c(1,2))
logcounts <- cpm(y, log=TRUE)
boxplot(logcounts, xlab="", ylab="Log2 counts per million", las=2)
abline(h=median(logcounts), col="blue")
title("Unnormalized logCPMs")
logcountsNorm <- cpm(yNorm, log=TRUE)
boxplot(logcountsNorm, xlab="", ylab="Log2 counts per million", las=2)
abline(h=median(logcountsNorm), col="blue")
title("Normalized logCPMs")
dev.off()
```

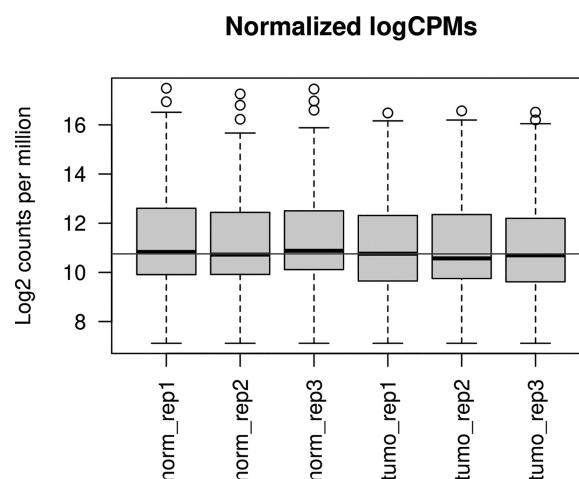


FIGURE 5.17 The boxplots of normalized log CPM.

Figure 5.17 shows the boxplot of the TMM-normalized data for each sample. The black line dividing each box represents the median of the count data, the top of the box shows the upper quartile, and the bottom of the box shows the lower quartile. The top and bottom whiskers show the highest and lowest count values, respectively, and the circles indicate the outliers.

We can use multidimensional scaling (MDS) [35] for representing relationships graphically between samples in multidimensional space and showing the overall differences between the gene expression profiles for the different samples. The MDS uses the pairwise dissimilarity Euclidean distances between samples in terms of the leading log-fold change (logFC) for the genes that best characterize the pair of samples. The leading logFC is calculated as the root-mean-square of the top log2-fold changes between the pair of samples (the default is 500 (top=500) logFCs). Edger uses “plotMDS” function to plot the MDS plot. The general syntax is as follows:

```
plotMDS(x, top=500, gene.selection = "pairwise", method = "logFC",
...)
```

The samples are then represented graphically in two dimensions such that the distance between points on the plot approximates their multivariate dissimilarity. The objects that are closer together on the MDS plot are more similar than the distant ones. In our example data, if there is a difference between the normal and tumor samples, then we can see clear patterns in multivariate dataset.

```
install.packages("RColorBrewer")
library(RColorBrewer)
png(file="MDSPplot.png")
pseudoCounts <- log2(yNorm$counts + 1)
colConditions <- brewer.pal(3, "Set2")
colConditions <- colConditions [match(sampleinfo$condition,
                                         levels(factor(sampleinfo$condition)))]
patients <- c(8, 15, 16) [match(sampleinfo$patient,
                                         levels(factor(sampleinfo$patient)))]
plotMDS(pseudoCounts, pch = patients, col = colConditions, xlim =
c(-2,2))
legend("topright", lwd = 2, col = brewer.pal(3, "Set2") [1:2],
       legend = levels(factor(sampleinfo$condition)))
legend("bottomright", pch = c(8, 15, 16),
       legend = levels(factor(sampleinfo$patient)))
dev.off()
```

As shown in Figure 5.18, the patterns are very clear that the samples are clustered by the condition; normal samples are grouped together, and tumor samples are grouped together, which indicates that the difference between the two groups is much larger than the differences within groups and it is likely that the between-group difference is statistically

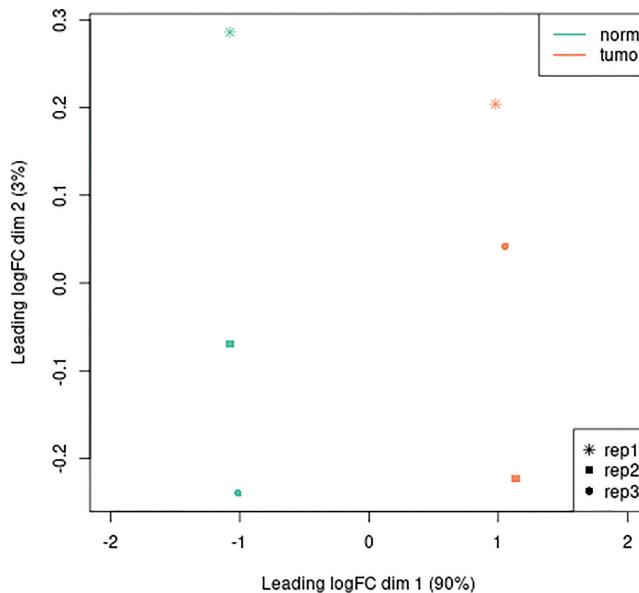


FIGURE 5.18 Multidimensional scaling (MDS) plot.

significant. The distance between the normal samples and the tumor sample is about 2 log2-fold change on the x -axis or 4 folds.

We can also use heatmap to cluster the most variable genes in the samples. We expect that some samples may have similar pattern depending on the given condition (normal or tumor). The following heatmap script will describe the relationships between samples using hierarchical clustering:

```
install.packages("gplots")
library("gplots")
png(file="heatmap1.png")
logcountsNorm <- cpm(yNorm, log=TRUE)
var_genes <- apply(logcountsNorm, 1, var)
select_var <- names(sort(var_genes, decreasing=TRUE)) [1:10]
highly_variable_lcpm <- logcountsNorm[select_var,]
mypalette <- brewer.pal(11,"RdYlBu")
morecols <- colorRampPalette(mypalette)
col.con <- c(rep("purple",3),
            rep("orange",3))[factor(sampleinfo$condition)]
heatmap.2(highly_variable_lcpm,
           col=rev(morecols(50)),trace="none",
           main="Top 10 most variable genes",
           ColSideColors=col.con,scale="row",
           margins=c(12,8),srtCol=45)
dev.off()
```

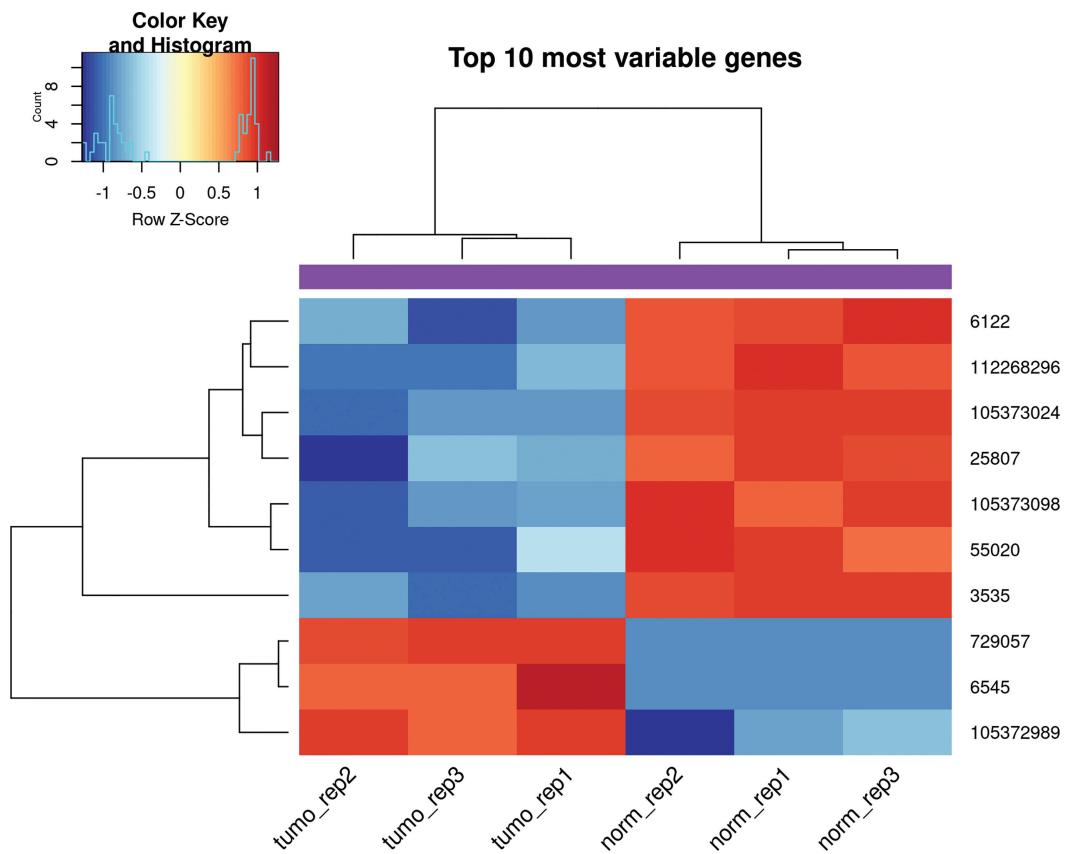


FIGURE 5.19 Heatmap clustering samples and top 10 variable genes.

Figure 5.19 shows that samples are clustered into tumor and normal samples based on the profiles of the genes in these samples.

5.3.7.8 Model Fitting

Once we have computed dispersion estimates, we can use them to fit the negative binomial generalized linear model, and then we can carry out the testing procedures for determining the differential expression. EdgeR has two functions to fit the RNA-Seq count data to the GLMs: the “glmQLFit” function which fits the data to a quasi-likelihood negative binomial generalized log-linear model and the “glmFit” function which fits the data to a negative binomial generalized log-linear model. The difference between the two GLM functions is that “glmQLFit” uses the trended negative binomial dispersion for fitting and then estimates the quasi-likelihood dispersion from the deviance, while “glmFit” uses the tagwise negative binomial dispersion for model fitting. You can use any one of them to fit the count data. Run the following to fit the count data to the quasi-likelihood negative binomial model:

```
fitq <- glmQLFit(yNorm, design)
names(fitq)
```

```
> fitq <- glmQLFit(yNorm, design)
names(fitq)
[1] "coefficients"           "fitted.values"          "deviance"
[4] "method"                  "counts"                  "unshrunken.coefficients"
[7] "df.residual"            "design"                 "offset"
[10] "dispersion"              "prior.count"           "AveLogCPM"
[13] "df.residual.zeros"      "df.prior"               "var.post"
[16] "var.prior"               "samples"                "genes"
```

FIGURE 5.20 Quasi-likelihood negative binomial model slots.

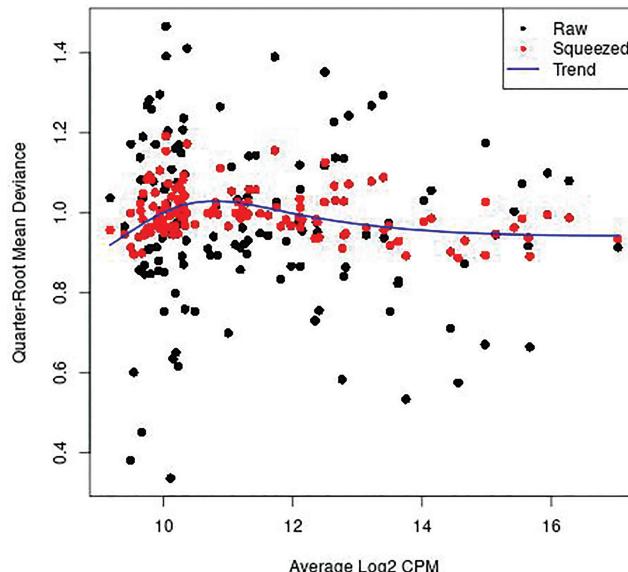


FIGURE 5.21 Quasi-likelihood dispersions plot.

Fitting the count data to a negative binomial model generates several data as shown in Figure 5.20. For instance, the GLM coefficients are in “coefficients” slot, fitted values are in “fitted.values” slot, and the estimated quasi-likelihood dispersions are in “dispersion” slot. The quasi-likelihood extends the negative binomial to account for gene-specific variability from both biological and technical aspects. We can visualize the quasi-likelihood dispersion with “plotQLDisp” function. The quasi-likelihood gene-wise dispersion estimates are squeezed toward a consensus trend, which will reduce the uncertainty of the estimates and improves testing power. The following script creates a quasi-likelihood dispersion plot showing the raw, squeezed, and trend dispersions (Figure 5.21):

```
jpeg('qlDispplots.jpg')
fitq <- glmQLFit(yNorm, design)
plotQLDisp(fitq, pch=16, cex=1.2)
dev.off()
```

Once we have fitted the count data to a GLM log-linear model, we can then be able to conduct the gene-wise statistical tests for a given coefficient (coef) or we can use “contrast” to

perform differential analysis. Since our primary goal is to study the difference between the normal and tumor samples, we can construct a contrast using “makeContrasts” function and then we can conduct the statistical test using the “glmQLFTest” function as follows:

```
my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq <- glmQLFit(yNorm, design)
qlfq<- glmQLFTTest(fitq, contrast=my.contrasts)
topTags(qlfq, n=10, adjust.method="BH", sort.by="PValue",
p.value=0.05)
```

The “qlfq” is a DGELRT object that stores the results of a GLM-based differential expression analysis for DGE data. The “topTags” function prints the top ten ($n=10$) set of the most significantly differential genes as shown in Figure 5.22. The p -value threshold is set to “ $p.value=0.05$ ” so only genes with p -value less than 0.05 will be listed. The negative log-fold changes (logFC) represent genes that are downregulated (down-expressed) in tumor samples over normal sample; logCPM is the log count-per-million; F is the test statistic for the null hypothesis that no difference in the gene expression between the normal and tumor samples; pvalue is the significance measure (p -value <0.05 is significant); and FDR is the false discovery rate.

To use GLM negative binomial model instead of the quasi-negative binomial model for the differential expression, you can use the following script:

```
> my.contrasts <- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq <- glmQLFit(yNorm, design)
qlfq<- glmQLFTTest(fitq, contrast=my.contrasts)
topTags(qlfq, n=10, adjust.method="BH", sort.by="PValue", p.value=0.05)
Coefficient: -1*conditionnorm 1*conditiontumo
      ENTREZID      SYMBOL      GENENAME
3535      3535      IGL      immunoglobulin lambda locus
84645     84645     C22orf23  chromosome 22 open reading frame 23
112268296 112268296 Loc112268296  uncharacterized LOC112268296
105373024 105373024 Loc105373024  uncharacterized LOC105373024
4627      4627      MYH9      myosin heavy chain 9
6122      6122      RPL3      ribosomal protein L3
402055    402055    SRRD      SRR1 domain containing
646023    646023    ADORA2A-AS1  ADORA2A antisense RNA 1
29781     29781     NCAPH2      non-SMC condensin II complex subunit H2
25807     25807     RHBDD3     rhomboid domain containing 3
      logFC      logCPM      F      PValue      FDR
3535     -4.989288  15.94189  1478.4742  6.829068e-16  9.014369e-14
84645     2.540331  15.42450  577.8426  5.257884e-13  3.470204e-11
112268296 -6.738806  12.86263  393.5526  7.684949e-12  3.381378e-10
105373024 -7.262517  11.97415  364.3041  1.312888e-11  4.332530e-10
4627      -3.170321  12.81681  321.7383  3.098015e-11  8.178761e-10
6122      -4.562492  12.10017  271.3722  9.975087e-11  2.194519e-09
402055    1.542530  14.55716  264.9553  1.174864e-10  2.215458e-09
646023    -2.610715  12.65725  160.2009  3.485719e-09  5.568814e-08
29781     1.217620  15.14121  155.7102  4.207486e-09  5.568814e-08
25807     -5.032951  11.42291  155.6470  4.218798e-09  5.568814e-08
```

FIGURE 5.22 The top ten significantly expressed genes.

```

my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitg<- glmFit(yNorm, design)
qlfg<- glmLRT(fitg, contrast=my.contrasts)
topTags(qlfg, n=10, adjust.method="BH", sort.by="PValue",
p.value=0.05)

```

Notice that we used contrast to conduct differential analysis to study the effect of tumor on the gene expression. However, the design may be more complex and EdgeR provides several ways to construct contrast based on the study design. For instance, you can add the intercept to the design matrix so the first column on the design will be the reference. We can also use “coef=” with “glmQLFTest” or “glmLRT” functions to indicate that the specified coefficients are zero. In more complex design like factorial design, an ANOVA-like analysis can be conducted. Refer to Edger users’ manual for more information about the designs.

EdgeR also provides “glmTreat” function to conduct gene-wise statistical tests for a given coefficient or a contrast relative to a specified fold-change threshold. For instance, assume that we need to test the hypothesis that the gene expression in normal cells and tumor cells is equal at a threshold of 2 log-fold change (lfc=2), assuming that a gene with a log-fold change less than 2 is equally expressed in both normal and tumor cells.

```

my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq <- glmQLFit(yNorm, design)
qlfq<- glmTreat(fitq, contrast=my.contrasts, lfc=2)
topTags(qlfq, n=10, adjust.method="BH", sort.by="PValue",
p.value=0.05)

```

Figure 5.23 shows the top ten significantly expressed genes using a log-fold change threshold (lfc=2).

Since the FDR (the rate of incorrectly rejecting null hypothesis) may be inflated in the case of multiple testing, it can be corrected using Benjamini–Hochberg method [36], which orders hypotheses and then rejects or accepts a hypothesis based on the Benjamini–Hochberg critical value.

The “decideTestsDGE” function can be used to display the total number of differentially expressed genes identified at an FDR of 5% (upregulated, downregulated, and no-significant regulation).

```

my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)
qlfq<-glmQLFTest(fitq, contrast=my.contrasts)
DEGenes<-decideTestsDGE(qlfq,
                        adjust.method="BH", p.value=0.05, lfc=2)
summary(DEGenes)

```

```

> my.contrasts <- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq <- glmQLFit(yNorm, design)
qlfq<- glmTreat(fitq, contrast=my.contrasts, lfc=2)
topTags(qlfq, n=10, adjust.method="BH", sort.by="PValue", p.value=0.05)
Coefficient: -1*conditionnorm 1*conditiontumo
  ENTREZID      SYMBOL          GENENAME
3535      3535      IGL      immunoglobulin lambda locus
112268296 112268296 LOC112268296 uncharacterized LOC112268296
105373024 105373024 LOC105373024 uncharacterized LOC105373024
6122      6122      RPL3      ribosomal protein L3
25807     25807     RHBDD3  rhomboid domain containing 3
4627      4627      MYH9      myosin heavy chain 9
105372989 105372989 LOC105372989 uncharacterized LOC105372989
105373098 105373098 KLHDC7B-DT  KLHDC7B divergent transcript
729057    729057    TOMM40P2  TOMM40 pseudogene 2
266623    266623    SLC25A5P1  solute carrier family 25 member 5 pseudogene 1
          logFC  unshrunk.logFC  logCPM      Pvalue      FDR
3535      -4.989288 -4.992432e+00 15.941889 9.523141e-14 1.257055e-11
112268296 -6.738806 -6.830298e+00 12.862631 5.122418e-10 3.380796e-08
105373024 -7.262517 -7.521916e+00 11.974146 8.437674e-10 3.712577e-08
6122      -4.562492 -4.596333e+00 12.100173 4.848134e-08 1.599884e-06
25807     -5.032951 -5.110081e+00 11.422915 8.362427e-07 2.207681e-05
4627      -3.170321 -3.178006e+00 12.816813 1.388883e-06 3.055542e-05
105372989 5.109566  5.205346e+00 11.322198 2.385231e-06 4.497865e-05
105373098 -5.478857 -5.734452e+00 10.309308 6.081516e-06 9.156425e-05
729057    7.733514  1.442695e+08 10.107829 6.243017e-06 9.156425e-05
266623    -7.153902 -1.442695e+08 9.491161 8.446801e-06 1.114978e-04

```

FIGURE 5.23 Differential analysis with log-fold change threshold.

```

> my.contrasts <- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)
qlfq<-glmQLFTest(fitq, contrast=my.contrasts)
DEGenes<-decideTestsDGE(qlfq,
                        adjust.method="BH", p.value=0.05, lfc=2)
summary(DEGenes)
Coefficient: -1*conditionnorm 1*conditiontumo
Down      23
NotSig   94
Up      15

```

FIGURE 5.24 The number of downregulated, upregulated, and nonsignificant genes.

Figure 5.24 shows the number of downregulated, nonsignificant, and upregulated genes at the threshold of 2 log-fold change and 0.05 significance level.

The level of the differential gene expression change can be visualized with the mean-difference plot (MD plot):

```

jpeg('mdPlotfitted.jpg')
my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)
qlfq<-glmQLFTest(fitq, contrast=my.contrasts)
DEGenes<-decideTestsDGE(qlfq,
                        adjust.method="BH", p.value=0.05, lfc=2)
plotMD(qlfq, status=DEGenes,

```

```

values=c(1,0,-1),
col=c("red","black","blue"),
legend="topright")
dev.off()

```

The MD plot in Figure 5.25 shows upregulated (red color), downregulated (blue color), and nonsignificant gene (black color) in log-fold change versus average log cpm (abundance) plain. The points are usually dense when all chromosomes are investigated. Remember that for the sake of simplicity, we demonstrate the differential analysis of gene expression in a single chromosome (chromosome 22). Both upregulated and downregulated genes are affected by the condition studied. That suggests that they may have a role on the condition or the effect may be due to the condition. In this example case, the upregulated and downregulated genes may have implications for the breast cancer. These genes can be singled out and studied to know their roles in the condition.

We can also use the volcano plots to display the differential gene expression results. A volcano plot is a scatterplot that shows the *p*-values versus the fold changes in gene expression. It enables a quick visual identification of genes with large fold changes that are also statistically significant. In a volcano plot, the upregulated genes are shown toward the right, the downregulated genes are shown toward the left, and the most statistically

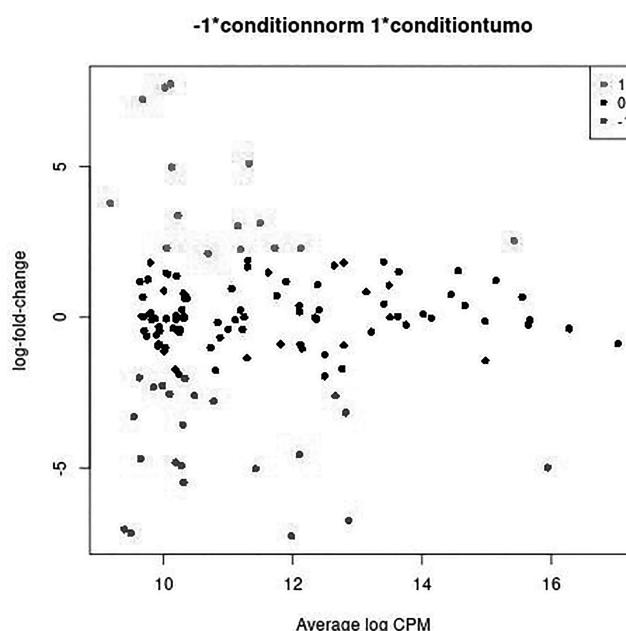


FIGURE 5.25 MD plot showing upregulated (red), downregulated (blue), and nonsignificant (black).

significant genes are toward the top of each side. We can create a volcano plot of the RNA-Seq results using the data of the top differentially expressed genes as shown in Figure 5.26.

```
#####
#volcano plot
jpeg('volcano.jpg')
fitq <- glmQLFit(yNorm, design)
qlfq<- glmTreat(fitq, contrast=my.contrasts, lfc=2)
resFilt<- topTags(qlfq, n=100, adjust.method="BH", sort.
by="PValue", p.value=1)
volcanoData <- cbind(resFilt$table$logFC,
-log2(resFilt$table$PValue))
colnames(volcanoData) <- c("logFC", "negLogPval")
plot(volcanoData, pch=19)
dev.off()
```

Once again, when all chromosomes are studied, the point distribution on the volcano plot will be very dense. Moreover, some programs are able to color the upregulated and downregulated genes for better visualization.

The heatmap has been known as the most popular graphical representation for visualizing complex and multidimensional data. We can use the heatmap to visualize differential expression of the genes. First, we need to use “cpm” function to convert read abundance into log2 CPM values. The heatmap can be created for the top differentially expressed genes. The following script creates a heatmap for the top 20 differentially expressed genes:

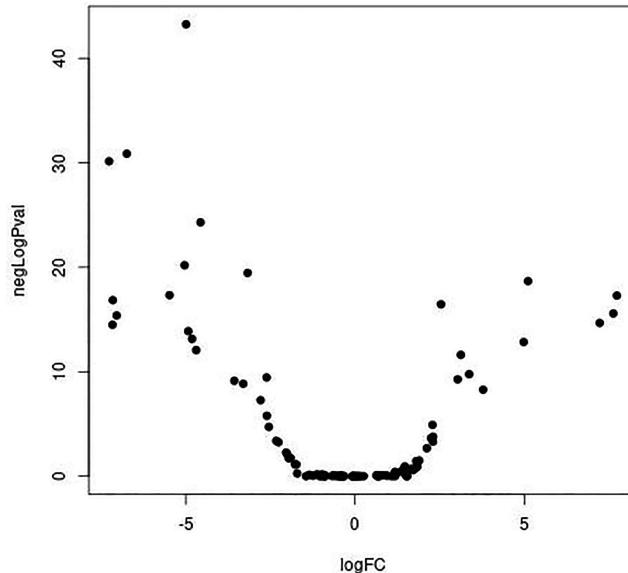


FIGURE 5.26 Volcano plot showing regulated genes.

```

jpeg('heatmap2.jpg')
my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)
qlfq<-glmQLFTest(fitq, contrast=my.contrasts)
DEGenes<-decideTestsDGE(qlfq,
    adjust.method="BH", p.value=0.05, lfc=2)
logCPM <- cpm(yNorm, prior.count=2, log=TRUE)
rownames(logCPM) <- yNorm$genes$SYMBOL
colnames(logCPM) <- paste(yNorm$samples$group, 1:3, sep="-")
o <- order(qlfq$table$PValue)
logCPM <- logCPM[o[1:20],]
logCPM <- t(scale(t(logCPM)))
col.pan <- colorpanel(100, "blue", "white", "red")
heatmap.2(logCPM, col=col.pan, Rowv=TRUE, scale="none",
    trace="none", dendrogram="both", cexRow=1, cexCol=1.4,
    margin=c(10,9), lhei=c(2,10), lwid=c(2,6))
dev.off()

```

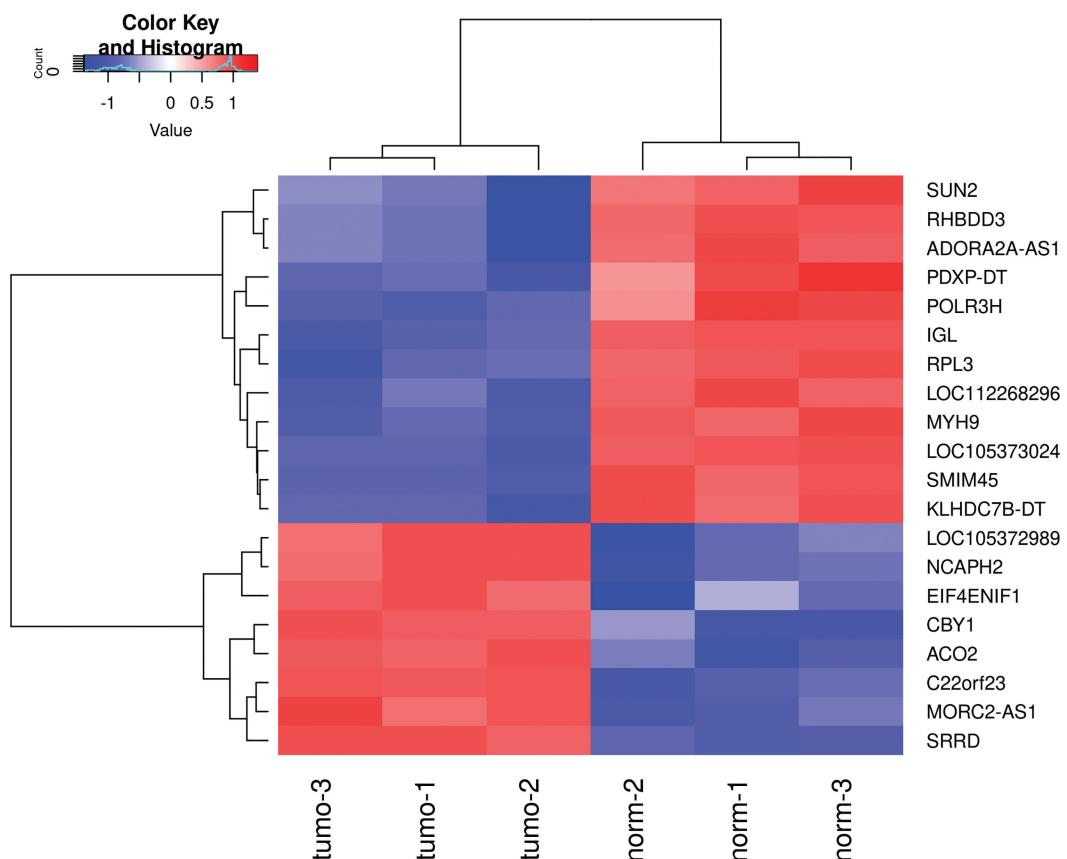


FIGURE 5.27 Heatmap for the top 20 most differentially expressed genes across the samples.

As shown in Figure 5.27, the heatmap clusters genes and samples based on Euclidean distance between the expression values. As expected, samples from the same group are clustered together.

5.3.7.9 Ontology and Pathways

After identifying the differentially expressed genes, the next step is to study the functions of these genes, their pathways, and the conditions associated with them based on the accumulated knowledge that already we have from previous studies and discoveries. This knowledge is available in databases like GO [37] and KEGG [38] databases as well as other pathway databases. Many of the genes associated with given biological processes are differentially expressed in a given condition like diseases. By performing GO analysis, we will be able to identify those biological processes, cellular locations, and molecular functions that are impacted by the condition studied. GO attempts to capture three aspects of the gene: (i) Biological processes (BP) that the gene may involve in, (ii) Molecular functions (MF), and (iii) Cellular components (CC), where the biological processes and molecular activities take place in the cell. It is important to know that GO terms aim to describe the normal functions, processes, or locations that gene products are involved in. It does not capture pathological processes, experimental conditions, or temporal information. Given a set of differentially expressed genes (upregulated or downregulated genes), GO and KEGG analyses will identify the GO terms and pathways, respectively, for each gene. EdgeR uses “goana” function for GO analysis and “kegga” function for KEGG analysis. Both functions require a DGELRT object and Entrez Gene identifiers (IDs) to annotate the genes. The NCBI Entrez Gene IDs must be present as row name as we did above. Also, it is important to specify the species studied. The following script performs GO analysis and annotates the significantly expressed genes (downregulated and upregulated genes) with the GO terms:

1	Term	Ont	N	Up	Down	P.Up	P.Down
2	GO:0031984 organelle subcompartment	CC	8	3	0	0.01282012	1
3	GO:0098791 Golgi subcompartment	CC	4	2	0	0.026609521	1
4	GO:0044431 Golgi apparatus part	CC	6	2	0	0.061478862	1
5	GO:0000009 alpha-1,6-mannosyltransferase activity	MF	1	1	0	0.073170732	1
6	GO:1990822 basic amino acid transmembrane transport	BP	1	1	0	0.073170732	1
7	GO:0015174 basic amino acid transmembrane transporter activity	MF	1	1	0	0.073170732	1
8	GO:0015802 basic amino acid transport	BP	1	1	0	0.073170732	1
9	GO:0008013 beta-catenin binding	MF	1	1	0	0.073170732	1
10	GO:0055007 cardiac muscle cell differentiation	BP	1	1	0	0.073170732	1
11	GO:0048738 cardiac muscle tissue development	BP	1	1	0	0.073170732	1
12	GO:0035051 cardiocyte differentiation	BP	1	1	0	0.073170732	1
13	GO:1905114 cell surface receptor signaling pathway involved in cell-cell signaling	BP	1	1	0	0.073170732	1
14	GO:0198738 cell-cell signaling by wnt	BP	1	1	0	0.073170732	1
15	GO:1905349 ciliary transition zone assembly	BP	1	1	0	0.073170732	1
16	GO:0004896 cytokine receptor activity	MF	1	1	0	0.073170732	1
17	GO:0019221 cytokine-mediated signaling pathway	BP	1	1	0	0.073170732	1
18	GO:0052917 dol-P-Man:Man(7)GlcNAc(2)-PP-Dol alpha-1,6-mannosyltransferase activity	MF	1	1	0	0.073170732	1
19	GO:0006488 dolichol-linked oligosaccharide biosynthetic process	BP	1	1	0	0.073170732	1
20	GO:0052824 dolichyl-pyrophosphate Man7GlcNAc2 alpha-1,6-mannosyltransferase activity	MF	1	1	0	0.073170732	1
21	GO:0009101 glycoprotein biosynthetic process	BP	1	1	0	0.073170732	1

FIGURE 5.28 Ontology annotation of the significantly expressed genes.

```

my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)
qlfq<-glmTreat(fitq, contrast=my.contrasts, lfc=2)
go <- goana(qlfq, species="Hs")
topGO20<-topGO(go, sort="up", n=20)
write.csv(topGO20, file="topGO20.csv")

```

Figure 5.28 shows GO IDs, terms, ontology (ONT), the total number of genes annotated with each ontology term (N), the number of genes that are significantly upregulated (up) and downregulated (down), the gene counts of the top significant GO ordered by the significance of the *p*-value, and the *p*-values for the upregulated (P.Up) and *p*-values for the downregulated (P.Down). Since the *p*-values are not adjusted for multiple testing, it is recommended to ignore GO terms with *p*-values greater than about 10^{-5} .

Since this exercise is based on a single chromosome (chromosome 22), we do not expect much information as when we analyze the entire genome. In general, GO analysis tells us about the different biological processes, their localizations in the cells, and molecular functions based on the upregulated and downregulated genes.

In the same way, we can perform KEGG pathway analysis to identify the molecular pathways and disease signatures (Figure 5.29).

```

my.contrasts
<- makeContrasts(conditiontumo-conditionnorm, levels=design)
fitq<-glmQLFit(yNorm, design)

```

1	Pathway	N	Up	Down	P.Up	P.Down	
2	path:hsa00510	N-Glycan biosynthesis	1	1	0	0.073170732	1
3	path:hsa00513	Various types of N-glycan biosynthesis	1	1	0	0.073170732	1
4	path:hsa04310	Wnt signaling pathway	1	1	0	0.073170732	1
5	path:hsa01100	Metabolic pathways	4	1	0	0.264990863	1
6	path:hsa01210	2-Oxocarboxylic acid metabolism	1	0	0		1
7	path:hsa04261	Adrenergic signaling in cardiomyocytes	1	0	0	1	1
8	path:hsa04925	Aldosterone synthesis and secretion	1	0	0	1	1
9	path:hsa05031	Amphetamine addiction	1	0	0	1	1
10	path:hsa04215	Apoptosis - multiple species	1	0	0	1	1
11	path:hsa01230	Biosynthesis of amino acids	1	0	0	1	1
12	path:hsa05219	Bladder cancer	1	0	0	1	1
13	path:hsa01200	Carbon metabolism	1	0	0	1	1
14	path:hsa05230	Central carbon metabolism in cancer	1	0	0	1	1
15	path:hsa04022	cGMP-PKG signaling pathway	1	0	0	1	1
16	path:hsa05207	Chemical carcinogenesis - receptor activation	1	0	0	1	1
17	path:hsa04725	Cholinergic synapse	1	0	0	1	1
18	path:hsa05220	Chronic myeloid leukemia	1	0	1	1	0.1300813
19	path:hsa00020	Citrate cycle (TCA cycle)	1	0	0	1	1
20	path:hsa05030	Cocaine addiction	1	0	0	1	1
21	path:hsa05171	Coronavirus disease - COVID-19	1	0	1	1	0.1300813

FIGURE 5.29 KEGG annotation of the significantly expressed genes.

```
qlfq<-glmTreat(fitq,contrast=my.contrasts, lfc=2)
keg <- kegga(qlfq, species="Hs")
keg20<- topKEGG(keg, sort="up", n=20)
write.csv(keg20,file="keg20.csv")
```

5.3.8 Visualizing RNA-Seq Data

We have already discussed some methods for visualizing the RNA-Seq data. For publication purpose, we can create high-resolution colored graphics using “*vidger*” Bioconductor package, which is meant to generate information-rich visualizations for the interpretation of differential gene expression results from edgeR, DESeq2, and cuffdiff [39]. The “*vidger*” package can be installed in R using the following:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("vidger")
Once the package has been installed, we can load it using:
library("vidger")
```

In the following, we will use “*vidger*” package to create plots to visualize the example RNA-Seq data. Open R and make the “features” directory where you saved the RNA-Seq count data file as your working directory. The *vidger* functions require DGEList object with group and normalized count data. The following script will create a DGEList, “*yNorm*” that can be used as input for the functions:

```
#Loading packages
library(edgeR)
library("vidger")
library(org.Hs.eg.db)
#Loading data
seqdata <- read.delim("htcount2.txt", stringsAsFactors=FALSE)
sampleinfo <- read.delim("sampleinfo.txt", stringsAsFactors=FALSE)
#Preparing data
countdata0 <- seqdata[,-(1:2)]
rownames(countdata0) <- seqdata[,1]
colnames(countdata0) <- sampleinfo$sampleid
countdata <- countdata0[rowSums(countdata0[])>0,]
group = factor(sampleinfo$condition)
#Creating DGEList object
y <- DGEList(countdata, group=group)
#Adding annotation
ENTREZID <- mapIds(org.Hs.eg.db, rownames(y),
                     keytype="SYMBOL", column="ENTREZID")
rownames(y$counts) <- ENTREZID
ann<-select(org.Hs.eg.db, keys=rownames(y$counts),
            columns=c("ENTREZID", "SYMBOL", "GENENAME"))
y$genes <- ann
```

```
#Removing rows without Entrez ids
i <- is.na(y$genes$ENTREZID)
y <- y[!i, ]
#Creating the design matrix
condition <- factor(sampleinfo$condition)
design <- model.matrix(~ 0 + condition)
#Filtering genes with low abundance
keep <- filterByExpr(y, design)
y <- y[keep, , keep.lib.sizes=FALSE]
# Normalizing count data
yNorm <- calcNormFactors(y)
#Estimating dispersions:
yNorm <- estimateDisp(yNorm, design)
```

Once you have run the above script successfully without any error, then you can use the “`vidger`” functions to create plots as follows.

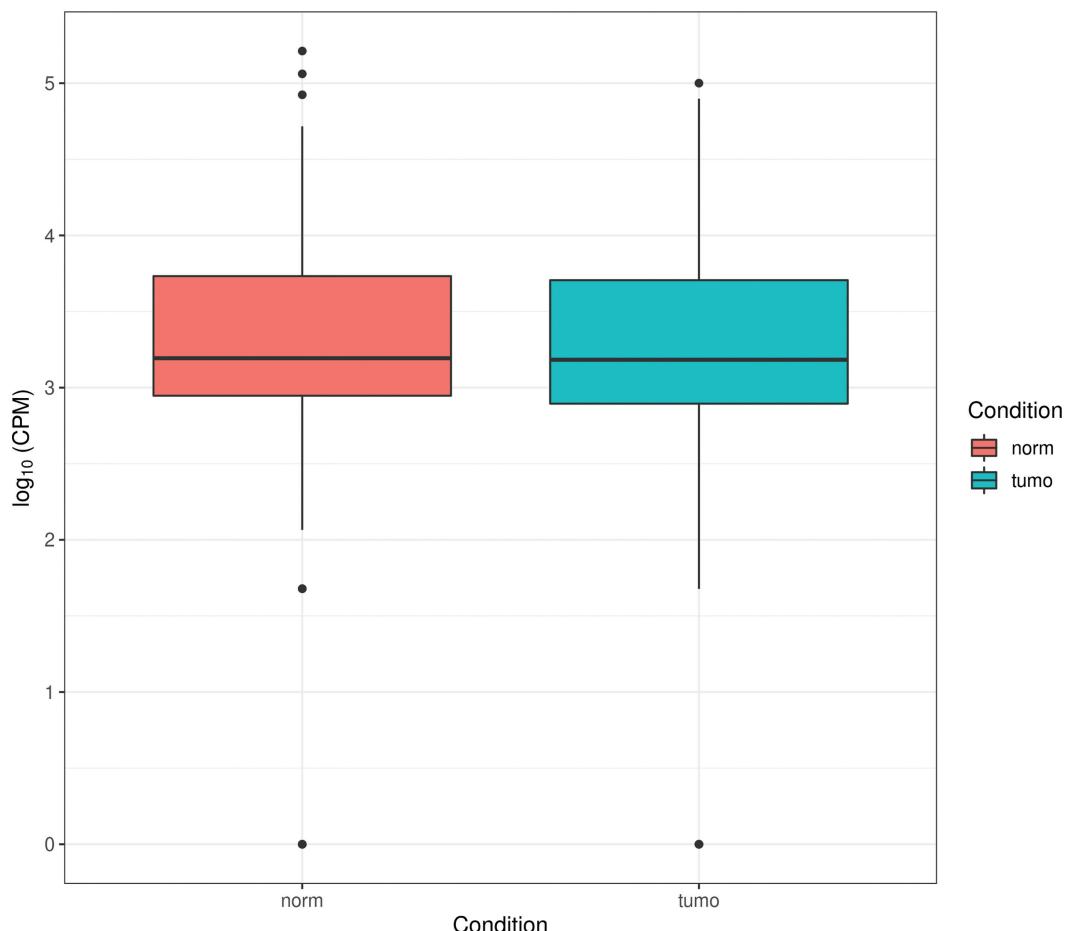


FIGURE 5.30 Box plots showing the distribution of normal and tumor counts in CPM.

5.3.8.1 Visualizing Distribution with Boxplots

We can use “vsBoxPlot()” function to create box plots to study the distribution of the count data based on condition studied. The count data can be FPKM or CPM normalized values. The function uses the DGEList object and creates boxplots for the log10 of either FPKM or CPM (Figure 5.30).

```
#Creating box plot
jpeg('boxplot.jpg')
vsBoxPlot(
  data = yNorm,
  d.factor = NULL,
  type = 'edger',
  title = "Box plots of normal and tumor count data",
  legend = TRUE, grid = TRUE)
```

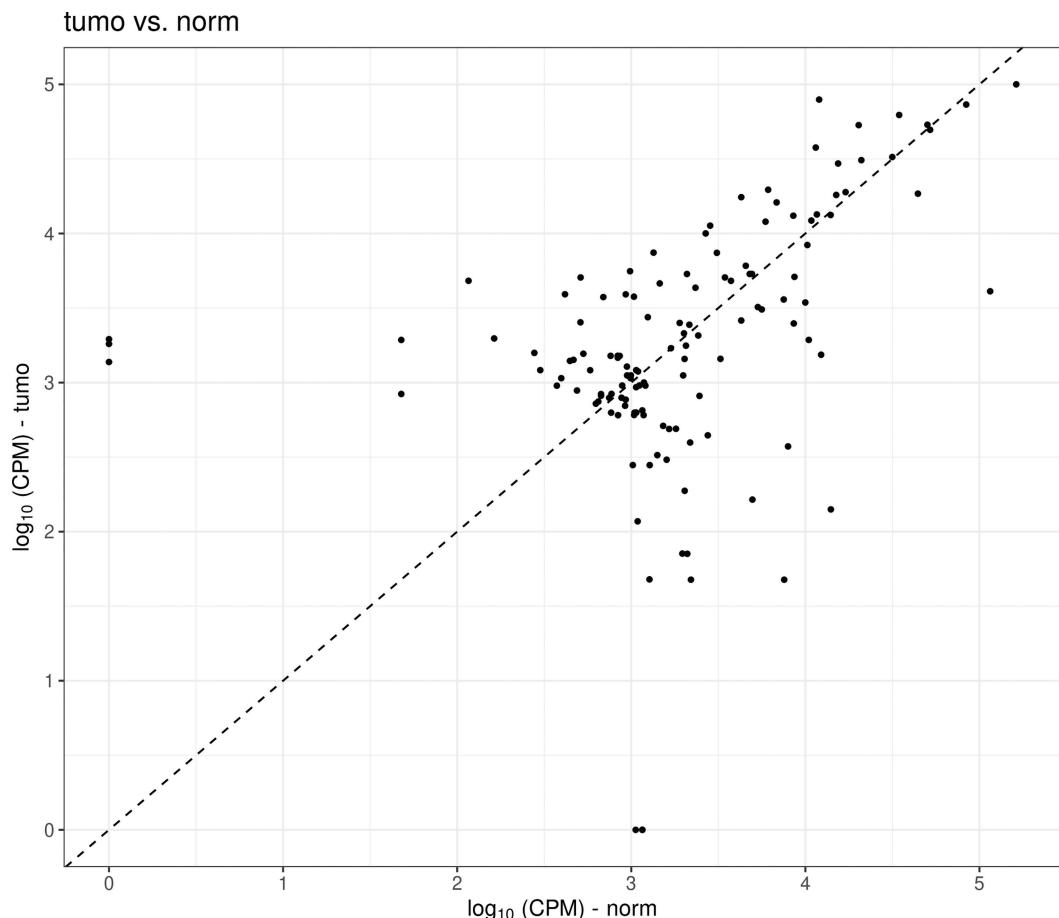


FIGURE 5.31 A scatter plot showing comparisons of log10 values of CPM for the two conditions.

```
dev.off()
```

5.3.8.2 Scatter Plot

The “vsScatterPlot()” function can be used to create a scatter plot that visualizes the comparisons of \log_{10} values of either FPKM or CPM measurements under the two conditions (Figure 5.31).

```
#Scatter plot
jpeg('CPMscatterplot.jpg')
vsScatterPlot(
  x = 'norm',
  y = 'tumo',
```

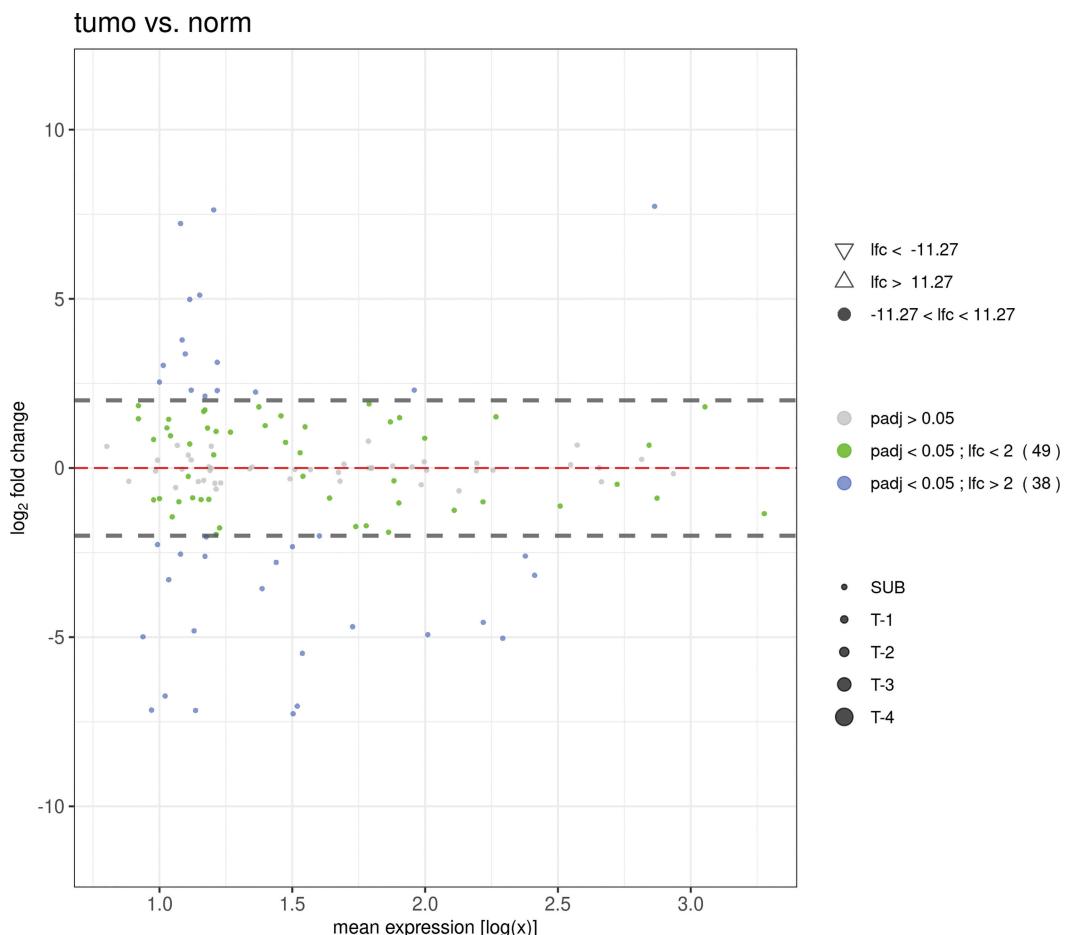


FIGURE 5.32 MA plot showing the log-fold change against mean expression.

```

data = yNorm,
type = 'edger',
d.factor = NULL,
title = TRUE, grid = TRUE)
dev.off()

```

5.3.8.3 Mean-Average Plot (MA Plot)

The “vsMAPlot()” function graphs the log-fold change versus the mean count to visualize the variance between the two conditions in terms of gene expression values.

```

jpeg('MAPlot.jpg')
vsMAPlot(
  x = 'norm', y = 'tumo',
  data = yNorm,
  d.factor = NULL,

```

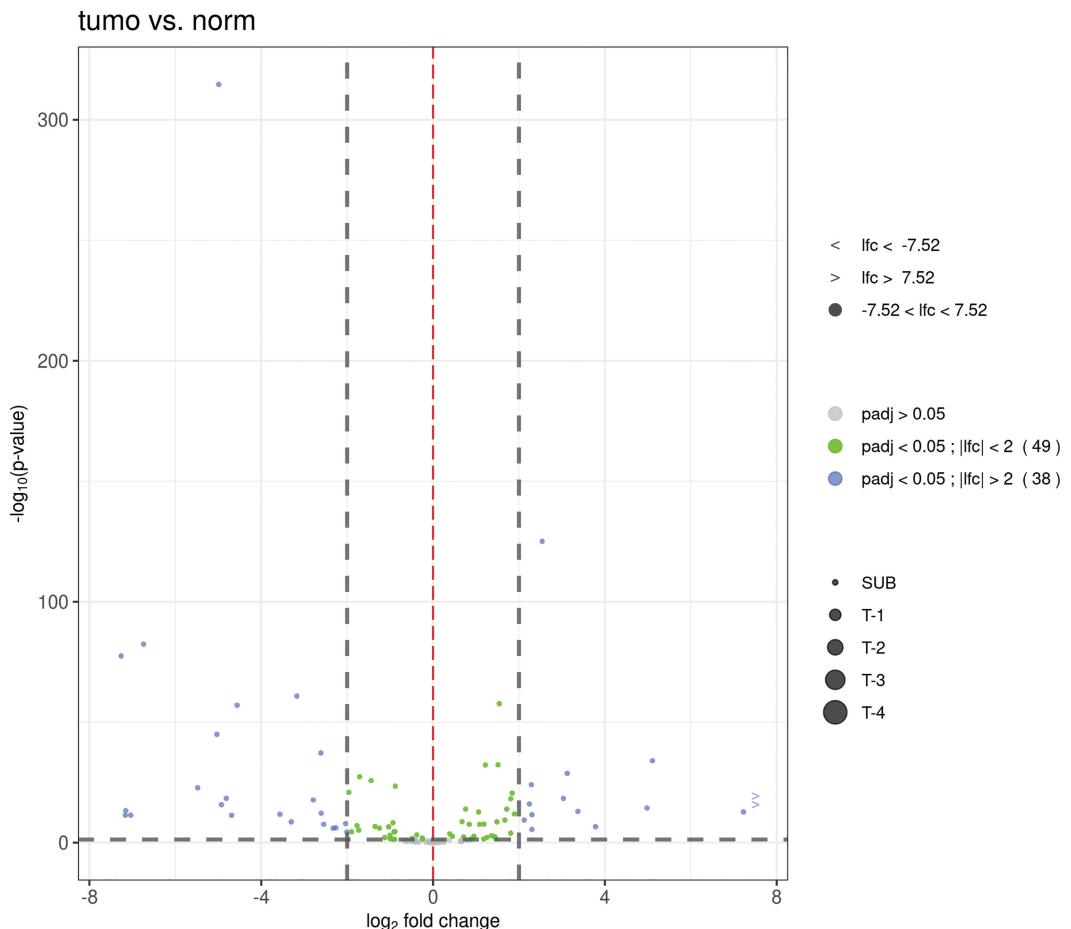


FIGURE 5.33 Volcano plot showing differential gene expression.

```

type = 'edger',
padj = 0.05,
y.lim = NULL,
lfc = 2,
title = TRUE, legend = TRUE, grid = TRUE)
dev.off()

```

As shown in Figure 5.32, the blue points on the graph represent the genes with statistically significant log-fold changes above the specified fold change indicated in the graph by the dashed lines. The green points represent the genes with statistically significant log-fold changes less than the specified fold change. The gray points represent the genes without statistically significant log-fold changes. Moreover, the values in parentheses for each legend color show the number of genes that meet the conditions. The triangular shapes represent values that are not displayed. The point size indicates the magnitude of the log-fold change.

5.3.8.4 Volcano Plots

As discussed above, the volcano plot is used to visualize the differential gene expression between two conditions. The “vsVolcano()” function graphs $-\log_{10}$ of p -values in the y-axis against the log2-fold changes in the x-axis (Figure 5.33).

```

jpeg('volcanoPlot2.jpg')
vsVolcano(
  x = 'norm', y = 'tumo',
  data = yNorm,
  d.factor = NULL,
  type = 'edger',
  padj = 0.05,
  x.lim = NULL,
  lfc = 2,
  title = TRUE,
  legend = TRUE, grid = TRUE,
  data.return = FALSE)
dev.off()

```

5.4 SUMMARY

The PCR allows studying gene expression in very narrow scale since only expression of few known genes can be investigated. Microarrays were emerged as the first technique for studying the gene expression of massive number of genes. However, recently, the RNA-Seq seems to replace microarrays because it is better at detecting gene transcripts and gene isoforms and it also provides more accurate and sensitive results for differential gene expression.

Studying gene expression allows researchers to identify the roles of genes in the biological activities in the cells and their implications for diseases. This specifically provides broad insight when massively parallel sequencing is used in the investigation of gene expression

across the whole genome. Genes code for proteins that determine traits and biological functions. RNA-Seq allows investigation of the entire transcriptome through the profiling of genes so researchers will know which sets of genes are coregulated during the conditions studied. RNA-Seq is used to study diseases like cancers and patient's response to therapeutics and other diseases.

The first steps of the workflow of RNA-Seq are similar to the steps of the other NGS applications as discussed in the previous chapters. The raw data is obtained in FASTQ files, which can be single end or paired end. The raw data is subjected to quality control for the trimming of adaptors and removal of duplicate sequences and low-quality reads. The cleaned raw data is either aligned to a reference genome or a reference transcriptome. The alignment program should be chosen depending on the read length and splicing awareness. STAR is a good choice for aligning short RNA-Seq reads. However, it requires sufficient memory and storage space for the indexing of the reference sequence and read alignment. The alignment information is stored in SAM/BAM files. Programs like Samtools and PICARD can be used to manipulate and generate alignment statistics from the BAM files. Quantification of reads aligned to each gene is essential in the RNA-Seq data analysis because it is the only way to see the gene transcript abundance, which is an indication for its biological activities. The reads aligned to each gene in BAM files are counted using reads counting programs like htseq-count and featureCount. In addition to the BAM files as inputs, these programs require also a reference annotation file in GTF format. The reads counts produced by the counting programs are stored in a tab-delimited file containing gene symbols or gene IDs and the corresponding counts of the aligned reads for each gene. The count file may contain a count column for each sample. Sample counts in different files can be merged for easy processing. The count data is usually normalized to be suitable for within-sample gene comparisons and between-sample gene comparisons. The normalization is made to adjust for the biases arising from the differences in the gene lengths, GC contents, and library sizes across samples. The normalization methods like RPKM/FPKM and CPM adjust for the biases arising from the difference in gene lengths; therefore, they can be used when comparison between the expressions of different genes within the sample is intended. Other normalization methods like TMM and RLE adjust for the bias arising from the difference in the library sizes across samples; therefore, they can be used for between-sample differential expression analysis. The count data generated by reads counting programs are further analyzed by the differential programs like edgeR, DESeq2, and cuffdiff. As an example, we used edgeR to demonstrate the steps of the differential analysis of the RNA-Seq data.

The differential analysis requires a metadata file that holds the information of the samples and the study design from which the design matrix is generated. Genes with low abundance can be filtered out since they do not contribute to the differential analysis. The normalized count data as response variable and the design matrix as explanatory variables are used for a GLM fitting. Since the count data is over-dispersed (variance is greater than the mean), the negative binomial distribution is assumed to fit the RNA-Seq count data to a log-linear model to estimate the dispersions and other statistics for the differential gene expression including log-fold change, *p*-values, and false discovery rates (FDR).

The expression level of a gene is described by the log-fold change of the gene transcript abundance with respect to a reference. This reference will be a different gene in the case of intra-sample comparison or the same gene in a different sample in the case of inter-sample differential analysis. The significance of differential gene expression is measured by the fold change, test statistic, *p*-value, adjusted *p*-value, and FDR. In the case of multiple comparison, the adjusted *p*-value is used. The expressed genes are usually sorted by *p*-values (from the lowest to the largest) so that it will be easy to identify the top upregulated and downregulated genes. These genes can be further analyzed by annotating them with their GO and KEGG pathways terms to gain insight into their biological interpretation under the conditions studied.

REFERENCES

1. Reyes JL, Kois P, Konforti BB, Konarska MM: The canonical GU dinucleotide at the 5' splice site is recognized by p220 of the U5 snRNP within the spliceosome. *Rna* 1996, 2(3):213–225.
2. Wilkinson ME, Charenton C, Nagai K: RNA Splicing by the spliceosome. *Annu Rev Biochem* 2020, 89: 359–388.
3. Haque A, Engel J, Teichmann SA, Lönnberg T: A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. *Genome Med* 2017, 9(1):75.
4. Vu TN, Deng W, Trac QT, Calza S, Hwang W, Pawitan Y: A fast detection of fusion genes from paired-end RNA-seq data. *BMC Genomics* 2018, 19(1):786.
5. Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M, Gingeras TR: STAR: ultrafast universal RNA-seq aligner. *Bioinformatics* 2013, 29(1):15–21.
6. Hoffmann S, Otto C, Kurtz S, Sharma CM, Khaitovich P, Vogel J, Stadler PF, Hackermüller J: Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Comput Biol* 2009, 5(9):e1000502.
7. Marco-Sola S, Sammeth M, Guigó R, Ribeca P: The GEM mapper: fast, accurate and versatile alignment by filtration. *Nat Methods* 2012, 9(12):1185–1188.
8. Li H, Durbin R: Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* 2010, 26(5):589–595.
9. Bushnell B: *BBMap: A Fast, Accurate, Splice-Aware Aligner*. In.: Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States); 2014.
10. Kukurba KR, Montgomery SB: RNA Sequencing and Analysis. *Cold Spring Harb Protoc* 2015, 2015(11):951–969.
11. Wu TD, Nacu S: Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics* 2010, 26(7):873–881.
12. Wei X, Wang X: A computational workflow to identify allele-specific expression and epigenetic modification in maize. *Genomics Proteomics Bioinfor* 2013, 11(4):247–252.
13. Grant GR, Farkas MH, Pizarro AD, Lahens NF, Schug J, Brunk BP, Stoeckert CJ, Hogenesch JB, Pierce EA: Comparative analysis of RNA-Seq alignment algorithms and the RNA-Seq unified mapper (RUM). *Bioinformatics* 2011, 27(18):2518–2528.
14. Kim D, Paggi JM, Park C, Bennett C, Salzberg SL: Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nat Biotechnol* 2019, 37(8):907–915.
15. Corchete LA, Rojas EA, Alonso-López D, De Las Rivas J, Gutiérrez NC, Burguillo FJ: Systematic comparison and assessment of RNA-seq procedures for gene expression quantitative analysis. *Sci Rep* 2020, 10(1): 19737.
16. Okonechnikov K, Conesa A, García-Alcalde F: Qualimap 2: advanced multi-sample quality control for high-throughput sequencing data. *Bioinformatics* 2016, 32(2):292–294.

17. DeLuca DS, Levin JZ, Sivachenko A, Fennell T, Nazaire MD, Williams C, Reich M, Winckler W, Getz G: RNA-SeQC: RNA-seq metrics for quality control and process optimization. *Bioinformatics* 2012, 28(11):1530–1532.
18. Wang L, Wang S, Li W: RSeQC: quality control of RNA-seq experiments. *Bioinformatics* 2012, 28(16):2184–2185.
19. Anders S, Pyl PT, Huber W: HTSeq--a Python framework to work with high-throughput sequencing data. *Bioinformatics* 2015, 31(2):166–169.
20. Liao Y, Smyth GK, Shi W: featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics* 2013, 30(7):923–930.
21. Robinson MD, Oshlack A: A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biol* 2010, 11(3):R25.
22. Benjamini Y, Speed TP: Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Res* 2012, 40(10):e72–e72.
23. Maza E: In Papyro Comparison of TMM (edgeR), RLE (DESeq2), and MRN Normalization Methods for a Simple Two-Conditions-Without-Replicates RNA-Seq Experimental Design. *Front Genet* 2016, 7:164.
24. Mortazavi A, Williams BA, McCue K, Schaeffer L, Wold B: Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nat Methods* 2008, 5(7):621–628.
25. Wagner GP, Kin K, Lynch VJ: Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. *Theory Biosci* 2012, 131(4):281–285.
26. Bushel PR, Ferguson SS, Ramaiahgari SC, Paules RS, Auerbach SS: Comparison of Normalization Methods for Analysis of TempO-Seq Targeted RNA Sequencing Data. *Front Genet* 2020, 11.
27. Robinson MD, Oshlack A: A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biol* 2010, 11(3):R25.
28. Anders S, Huber W: Differential expression analysis for sequence count data. *Genome Biol* 2010, 11(10):R106.
29. Bullard JH, Purdom E, Hansen KD, Dudoit S: Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC Bioinformatics* 2010, 11(1):94.
30. Finotello F, Di Camillo B: Measuring differential gene expression with RNA-seq: challenges and strategies for data analysis. *Brief Funct Genomics* 2015, 14(2):130–142.
31. Ver Hoef JM, Boveng PL: Quasi-Poisson vs. negative binomial regression: how should we model overdispersed count data? *Ecology* 2007, 88(11):2766–2772.
32. Law CW, Zeglinski K, Dong X, Alhamdoosh M, Smyth GK, Ritchie ME: A guide to creating design matrices for gene expression experiments. *F1000Res* 2020, 9: 1444.
33. Robinson MD, McCarthy DJ, Smyth GK: edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 2010, 26(1):139–140.
34. Carlson M: org.Hs.eg.db: Genome wide annotation for human. R package version 3100 2019.
35. Mead A: Review of the development of multidimensional scaling methods. *J R Stat Soc D* 1992, 41(1):27–39.
36. Benjamini Y, Hochberg Y: Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J R Stat Soc B* 1995, 57(1):289–300.
37. Consortium GO: The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res* 2004, 32(suppl_1):D258–D261.
38. Kanehisa M, Araki M, Goto S, Hattori M, Hirakawa M, Itoh M, Katayama T, Kawashima S, Okuda S, Tokimatsu T et al: KEGG for linking genomes to life and the environment. *Nucleic Acids Res* 2007, 36(suppl_1):D480–D484.
39. vidger: Create rapid visualizations of RNAseq data in R [https://bioconductor.org/packages/release/bioc/html/vidger.html]

Chromatin Immunoprecipitation Sequencing

6.1 INTRODUCTION TO CHROMATIN IMMUNOPRECIPITATION

In complex organisms like multicellular eukaryotes, each somatic cell in the body carries exactly the same genome (set of DNA molecules) but gene expression varies from a type of cells to another. A genome is formed of a set of chromosomes found in pairs in most eukaryotic cells. These cells are called diploids. A chromosome is a DNA molecule formed by the four basic nucleotides of the nucleobases: adenine (A), cytosine (C), guanine (G), and thymine (T). The lengths of chromosomes of an organism vary and each chromosome includes a different set of genes that code for different proteins. For the set of chromosomes to fit the nucleus, each chromosome includes multiple repeated units around 147 bp in length. These repeated units of DNA sequences wrapped around eight units (octamer) of histone proteins forming structures called nucleosomes. A histone octamer is formed of four different core histone units (H2A, H2B, H3, and H4). Nucleosomes are connected by other family of histones called linker histone H1, which is highly susceptible to modifications. A chromosome and packing proteins (histones) are together called chromatin, and when we mention a chromatin segment, we refer to a piece of raw DNA with histones.

Some environmental factors may lead to changes in chromatin causing the so-called epigenetic changes that involve covalent modifications to chromatin structure. These epigenetic chromatin modifications include DNA methylation, histone modification (acetylation, phosphorylation, methylation, and ubiquitylation), and chromatin remodeling. Such modifications affect gene expression without altering the DNA sequence and they determine the expression status of the protein-coding RNA (mRNAs) and other non-coding RNA molecules. In general, epigenetic changes include any process that alters gene expression without changing the DNA sequence and leads to modifications that can be transmitted to daughter cells. A wide variety of illnesses including cancers and many other health

conditions were found to be linked to epigenetic mechanisms. An epigenome consists of all epigenetic modifications, with the genome of an organism, that regulate the activity (expression) of the genes; these modifications can be passed down to an organism's offspring [1]. Epigenomics is the study of the complete set of epigenetic modifications on the genome of an organism. Researchers use chromatin immunoprecipitation (ChIP) to examine the interactions between epigenetic components (proteins and DNA) and profiling of DNA methylations in their original context. ChIP are used to identify specific genes and sequences where a protein of interest binds, across the entire genome, providing critical information about their regulatory functions and mechanisms. The laboratory protocol of the ChIP includes fixation of in-vivo chromatin-bound proteins with formaldehyde to stabilize the protein on the chromatin, and then sonication or restriction enzymes are used to cut chromatin into short random fragments usually around 200 bp. ChIP can be performed without formaldehyde crosslinking by digesting chromatin with micrococcal nuclease, which is an enzyme that can break chromatin into the desired fragment size (Native ChIP). Antibodies specific for the proteins of interest are added to the short chromatin fragment. These enzymes form immunoprecipitated DNA–protein–antibody complexes that can be separated from the non-immunoprecipitated chromatin (DNA without protein of interest) using beads. The cross-linked formaldehyde is then removed either by heating or by digesting the protein component of the chromatin. In the final step, only the DNA fragments, to which proteins of interest were bound, are isolated and purified. Up to this step, we would have the DNA fragments that were the target for the epigenetic modification. The next step is to characterize these fragments by identifying the sequences of the binding sites and the affected genes and that provides important information about the binding sites of the transcription factors (TFs), function and regulation of the genes, and the impact of the activities of the genes on the condition studied.

6.2 CHIP SEQUENCING

Researchers use different techniques to characterize the ChIP purified DNA fragments. Northern blot, polymerase chain reaction (PCR), and microarray are some of the methods. However, only recently, sequencing these fragments with high-throughput methods has become the most commonly used and effective method for studying epigenetic modifications. The technique of sequencing the DNA fragments isolated from immunoprecipitation is called ChIP-Seq.

Highly specific antibodies to the targeted proteins are used for the ChIP-Seq so that only the DNA fragments affected by the epigenetic modifications are isolated. The ChIP-Seq also requires control DNA fragments extracted from the same samples but from the DNA regions that were not affected by the epigenetic modification (non-immunoprecipitated chromatin fragments) or the input DNA purified from the fragmented chromatin before the antibody incubation step. The control DNA serves as a baseline to normalize the ChIP data. Normalization with control DNA data can reduce false positives originated from biases that cause overrepresentation of reads. Possible source of bias includes the nonuniform fragmentation during sonication (sonication bias), PCR amplification which tends to over-amplify GC-rich regions (PCR bias), sequencing bias, and mapping bias.

The library preparation of the ChIP-Seq DNA fragments follows the same steps as that of the whole genome sequencing (WGS), which includes fragmentation, end repair, adaptor ligation, and enrichment. The sequencing steps follow the same steps used for DNA sequencing by the sequencing technology. The sequencing raw data includes millions of ChIP-Seq reads.

The sequencing strategies used for ChIP-Seq are the same as the ones followed for the WGS and RNA-Seq. The design of the ChIP-Seq experiment is usually tailored to the condition studies and that design will guide the subsequent data analysis. The raw data produced by the sequencer are raw reads in FASTQ files. Sequencing can be single end or paired end, short reads (e.g., Illumina) or long reads (e.g., PacBio). However, most ChIP-Seq datasets have been generated using single-end libraries and we should be aware that some programs do not use paired-end libraries. The run can be for a single sample or multiplexed for several samples; the fragments of each sample in the run are with a unique barcode.

6.3 CHIP-SEQ ANALYSIS WORKFLOW

In general, the ChIP-Seq analysis workflow includes raw data acquisition, quality control, read alignment, alignment quality control, peak calling, combining peak calls, and final analysis (visualization, motif discovery, and annotation and functional enrichment). You are already familiar with the first four steps, which were discussed in detail in Chapters 1 and 2. ChIP-Seq raw data can be either provided by the sequencing facility for an experiment or can be downloaded from a database. In either case, you may need to reprocess the FASTQ files (refer to Chapter 1). There are several databases from which we can download ChIP-Seq raw data submitted by other researchers either as supplementary material for their publications or may be submitted as part of a project dedicated for investigating some conditions and the data is made public for researchers. The NCBI SRA is the most commonly used database for these purposes and it integrates most of the other public databases. FASTQ files are downloaded from the NCBI SRA database using SRA-toolkit, which we used in the previous chapters. ChIP reads can be subjected to quality control by following the steps discussed in Chapter 1. FastQC program is used to assess the quality of the reads in the files. The reads then can be preprocessed, if needed, to remove the reads with low quality, trim the low-quality ends or adaptor sequence, and remove duplicate reads and other technical reads. The step of quality control is always crucial as in other sequencing applications to avoid misleading interpretation of results. Read mapping is performed by aligning ChIP reads and control reads to a reference genome. The same aligners (e.g., BWA and Bowtie) used for aligning reads from WGS can also be used for ChIP-Seq data (refer to Chapter 2). The alignment information produced by the aligner is stored in SAM/BAM files. Before proceeding to the next step, we can remove duplicate reads. Duplicate reads are generated from a single read; they are identical and aligned to the same region forming low library complexity in that region. For ChIP-Seq data, reads are aligned upstream and downstream around the binding sites, leaving the regions of the binding site with low sequence coverage. The read pileup density (also called signal) around the binding sites should form bimodal enrichment patterns, with Watson strand tags enriched upstream of binding and Crick strand tags enriched downstream. The shape

of the ChIP-Seq signal may vary depending on the binding protein studied. The ChIP-Seq signal can be sharp, broad, or a mix of sharp and broad signal. The sharp signal characterizes the binding site of the TF which binds to a specific site in the DNA sequence called motif. Histones form broad ChIP-Seq signals because they span several nucleosomes and may cover several nucleotides on the DNA. The RNA polymerase II (Pol II) initiates the process of transcription by localizing on the promoter region of the gene and then it moves during the messenger RNA transcription. Therefore, the ChIP-Seq signal of Poly II may include both sharp and broad signals (Figure 6.1).

Peak-calling programs use sliding windows to scan the genome for these patterns to locate the binding regions by counting both Watson and Crick tags. However, for these kinds of tags to fit in a single window, they must be shifted to the center so that Watson tags are shifted toward the 3' end and Crick tags are shifted toward the 5' end to form a peak in the putative binding site. Peak-calling programs like MACS take advantage of the bimodal pattern to empirically model the shifting size to precisely locate the binding sites [2] on the genomic DNA sequences.

Peak calling is a step unique to ChIP-Seq data analysis and it aims to identify the genomic regions occupied by the protein of interest and enriched due to the ChIP. The abundance of the aligned reads normalized by input reads in a sliding window is the basis of the peak calling, which is performed using statistics that determine peak significance. The ChIP-Seq tags are usually normalized by input read (control), but some peak callers can also call peaks without using input reads. Instead, they assume even background signal



FIGURE 6.1 Sharp signal (TF), broad signal (histones), and mixed signal (Poly II).

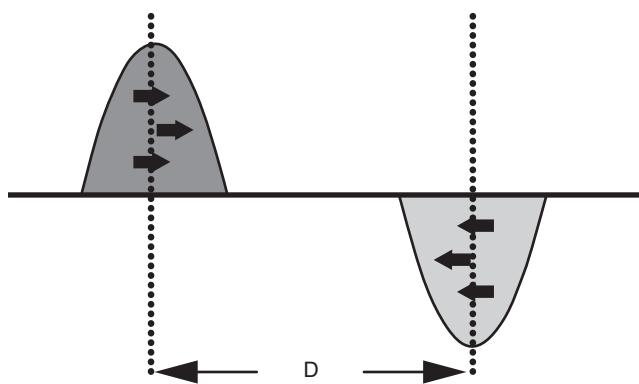


FIGURE 6.2 ChIP-Seq read alignment. (The peaks represent reads aligned to the reference genome.)

or other strategies. There are several peak-calling programs that use their own algorithms to define protein-binding sites in the genome by identifying regions where sequence reads are enriched after mapping to a reference genome. The peak caller assumes that ChIP-Seq reads should align in a larger number to the sites of protein binding than to other regions on the genome (Figure 6.2).

Peak-calling programs use different strategies to compute the statistical significance of peaks in the binding sites. Some peak callers assume Poisson or negative binomial distribution to model the counts of the reads and to compute the *p*-value for the statistical significance of the peak with respect to the background. Since multiple windows (thousands) are tested generating multiple *p*-values, the chance of making Type I error (false positive) will increase. Some peak callers adjust the *p*-value based on the number of windows by computing the false discovery rate (FDR). Other callers use the height of peaks over background without providing statistical significance metric and others use machine learning to generate statistical metrics that allow peak calling. Sequencing depth and library complexity are crucial for statistical significance of the fold enrichment.

In general, most peak aligners perform well. However, attention should be paid to the type of binding sites that a peak caller is good for. For instance, some callers are good for TFs (e.g., SISSRs [3]) and some are good for histone modifications, and some can handle both (e.g., MACS [2]). Some callers do not use paired-end library although paired-end reads can be treated as single-end reads by using either forward or reverse reads but not both. HOMER [4] caller was developed as a tool for *de novo* motif identification from peak regions. JAMM [5] requires replicated samples to improve confidence in peak calling. We should also pay attention to how a caller handles broad and sharp peaks. Callers may merge the close peaks and that may lead to the loss of some resolution. ChIP-Seq reads originated for histone modification generate a broad peak signal that requires a large region. However, determining a region boundary for the histone enrichment is still a challenge for the peak callers. In contrast to the histone modifications, ChIP-Seq signals of TFs and Poly II exhibit sharp peak signal, and therefore, the peak callers suitable for TFs and Poly II should be able to identify those narrow regions. In the following, we will discuss the steps of ChIP-Seq workflow with a worked example.

6.3.1 Downloading the Raw Data

For practicing with ChIP data, we will download data from ENCODE project at “<https://www.encodeproject.org>”. The raw data is from a ChIP-Seq experiment with an accession number ENCSR000EYL that includes three samples from HeLa-S3 cell line established from cervical adenocarcinoma of Henrietta Lacks, who was an African American woman died on October 4, 1951, at the age of 31, but her cells continue to have impact on the world by making significant contributions to the scientific progress and advances in human health. This ChIP experiment targeted DNA-directed RNA polymerase II subunit RPB1 (encoded by POLR2A gene), which is the largest subunit of RNA polymerase II that synthesizes all mRNA in eukaryotes. It initiates the transcription by allowing a single-stranded DNA template strand of the promoter of a targeted gene to position itself within its central active site. The mRNA is formed as the complementary transcript to the template

strand forming an initial DNA–RNA hybrid from which the new mRNA transcript is separated. The purpose of this exercise is to investigate the promoter regions in the gene targeted by the DNA-directed RNA polymerase II subunit RPB1 during gene transcription. The raw data consists of four single-end FASTQ files generated by Illumina Genome Analyzer and available at ENCODE database with the accession numbers: ENCFF000XJP, ENCFF000XJS, and ENCFF000XKD, and the accession number of the input data (control) is ENCSR000EZM. For the sake of keeping the files organized, we can create a project directory called “chipseq”, and inside that directory, we can create a subdirectory called “data” where we can download the FASTQ files as follows:

```
mkdir chipseq; cd chipseq; mkdir data
wget \
  -O "data/ENCFF000XJP_chp1.fastq.gz" \
  "https://www.encodeproject.org/files/ENCFF000XJP/@@download/
  ENCFF000XJP.fastq.gz"
wget \
  -O "data/ENCFF000XJS_chp2.fastq.gz" \
  "https://www.encodeproject.org/files/ENCFF000XJS/@@download/
  ENCFF000XJS.fastq.gz"
wget \
  -O "data/ENCFF000XKD_chp3.fastq.gz" \
  "https://www.encodeproject.org/files/ENCFF000XKD/@@download/
  ENCFF000XKD.fastq.gz"
wget \
  -O "data/ENCFF000XGP_inp0.fastq.gz" \
  "https://www.encodeproject.org/files/ENCFF000XGP/@@download/
  ENCFF000XGP.fastq.gz"
```

The four files will be downloaded into the “data” directory. The four files are ENCFF000XJP_chp1.fastq.gz, ENCFF000XJS_chp2.fastq.gz, ENCFF000XKD_chp3.fastq.gz, and ENCFF000XGP_inp0.fastq.gz. The latter is the FASTQ file that contains the input or control data.

6.3.2 Quality Control

The quality control is an important step in all sequencing data analysis workflows. The quality of the reads in the FASTQ file can be assessed by an appropriate program like FastQC to check the read quality, technical sequences such as adaptor dimer and PCR duplicate reads, GC-content bias, and other sequencing biases. We should try to fix any potential problem as possible before proceeding to the mapping step. Refer to Chapter 1 for the quality assessment metrics and the approaches to fix the potential faults.

```
cd data
fastqc \
  ENCFF000XJP_chp1.fastq.gz \
  ENCFF000XJS_chp2.fastq.gz \
```

```
ENCFF000XKD_chp3.fastq.gz \
ENCFF000XGP_inp0.fastq.gz
```

Then, we can display the reports in an Internet browser using Firefox command as follows:

```
firefox \
ENCFF000XJP_chp1_fastqc.html \
ENCFF000XJS_chp2_fastqc.html \
ENCFF000XKD_chp3_fastqc.html \
ENCFF000XGP_inp0_fastqc.html
cd ..
```

To avoid repeating what had been discussed in Chapter 1, we will assume that the four FASTQ files are cleaned and ready for the next step.

6.3.3 ChIP-Seq and Input Read Mapping

The second step in the ChIP-Seq data analysis, after data acquisition and quality control, is aligning both ChIP-Seq reads and input reads to a reference genome, following the same steps discussed in Chapter 2. You can use an aligner of your choice; however, in this example, we will use Bowtie2. First, we need to download the FASTA file of the current human reference genome from a reliable database such as NCBI and UCSC. We prefer the UCSC reference genome version because the chromosomes are given names instead of accession numbers. After downloading the compressed FASTQ file, it must be decompressed using “gunzip” and indexed with “samtools faidx” command. After the file is indexed with Samtools, we must use Bowtie2 to build an index for the reference FASTA file. The following commands, create the directory “ref” where the human reference genome is downloaded, decompressed, and indexed with both Samtools and Bowtie2. Refer to Chapter 2 for Samtools and Bowtie2 installation and uses.

```
mkdir ref; cd ref
wget https://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/hg19.
fa.gz
gunzip -d hg19.fa.gz
samtools faidx hg19.fa
bowtie2-build hg19.fa hg19
cd ..
```

Once the above operations have been performed successfully, we can use Bowtie2 to align both ChIP-Seq reads and control reads to the reference genome; since each file is aligned separately, four SAM files will be produced.

```
mkdir bam
bowtie2 \
-p 4 \
-x ref/hg19 \
```

```

-U data/ENCFF000XGP_inp0.fastq.gz \
-S bam/ENCFF000XGP_inp0.sam \
2> bam/inp0.log
bowtie2 \
-p 4 \
-x ref/hg19 \
-U data/ENCFF000XJP_chp1.fastq.gz \
-S bam/ENCFF000XJP_chp1.sam \
2> bam/chp1.log
bowtie2 \
-p 4 \
-x ref/hg19 \
-U data/ENCFF000XJS_chp2.fastq.gz \
-S bam/ENCFF000XJS_chp2.sam \
2> bam/chp2.log
bowtie2 \
-p 4 \
-x ref/hg19 \
-U data/ENCFF000XKD_chp3.fastq.gz \
-S bam/ENCFF000XKD_chp3.sam \
2> bam/chp3.log

```

The four SAM files produced by the above commands contain the alignment information of the reads. However, they may also include alignment information that we do not need and removing that will make us focus only on the regions of interest and also reduce the computational complexity. We can remove the mitochondrial read alignments, which are defined as “chrM” in the chromosome field of the SAM file and the unidentified, random, and haploid reads, which are defined as “chrUn”, “random”, and “*hap*”, respectively, keeping only the reads aligned to the human chromosomes. We can use “sed” Linux command to do that and the filtered alignments are saved in new files.

```

cd bam
sed '/chrM/d;/random/d;/chrUn/d;/hap/d' ENCFF000XGP_inp0.sam >
ENCFF000XGP_inp0_filt.sam
sed '/chrM/d;/random/d;/chrUn/d;/hap/d' ENCFF000XJP_chp1.sam >
ENCFF000XJP_chp1_filt.sam
sed '/chrM/d;/random/d;/chrUn/d;/hap/d' ENCFF000XJS_chp2.sam >
ENCFF000XJS_chp2_filt.sam
sed '/chrM/d;/random/d;/chrUn/d;/hap/d' ENCFF000XKD_chp3.sam >
ENCFF000XKD_chp3_filt.sam

```

We can then convert the SAM files into BAM files using “samtools view” command.

```

samtools view -S -b ENCFF000XGP_inp0_filt.sam > ENCFF000XGP_inp0_
filt.bam

```

```
samtools view -S -b ENCFF000XJP_chp1_filt.sam > ENCFF000XJP_chp1_filt.bam
samtools view -S -b ENCFF000XJS_chp2_filt.sam > ENCFF000XJS_chp2_filt.bam
samtools view -S -b ENCFF000XKD_chp3_filt.sam > ENCFF000XKD_chp3_filt.bam
```

The BAM file takes less storage space. Then, we can delete the SAM file to save some storage space if we need to. Just be careful not to delete the BAM files.

Now, we have three BAM files for the three ChIP-Seq data and one file for the control data. Before proceeding, we need to know the number of alignments in each file and then draw a sample of control reads approximately equal to the reads of any of the ChIP-Seq files to be the input reads for that ChIP-Seq file. We do that to avoid library coverage bias. The following “samtools view” commands count the alignments in each BAM file:

```
samtools view -c ENCFF000XGP_inp0_filt.bam
samtools view -c ENCFF000XJP_chp1_filt.bam
samtools view -c ENCFF000XJS_chp2_filt.bam
samtools view -c ENCFF000XKD_chp3_filt.bam
```

Table 6.1 shows the number of aligned reads in each BAM file and the factor, which is the read count of a ChIP-Seq file divided by the read count of the control file. This fraction is used to sample input reads from the control file for that ChIP-Seq file.

The following commands store the counts in bash variables and then use “samtools view” command to draw a subsample of reads from the control file and store them in a separate control file for that ChIP-Seq file. The “-b” option is to output a BAM file and “-s” option is to draw a subsample from the file.

```
inpc=$(samtools view -c ENCFF000XGP_inp0_filt.bam)
chp1=$(samtools view -c ENCFF000XJP_chp1_filt.bam)
fact1=$(echo "scale=6; $chp1/$inpc" | bc)
samtools view -b -s $fact1 ENCFF000XGP_inp0_filt.bam >
ENCFF000XGP_inp0_filt_inp1.bam
```

TABLE 6.1 Read Count in Each BAM File, the Fraction for Sampling Reads from the Control BAM file, and Number of Reads in the Control File for Each ChIP-Seq File

Sample	Read Count	Sampling Factor	Control Read Count
ENCFF000XGP_inp0_filt.bam	30,923,163	N/A	N/A
ENCFF000XJP_chp1_filt.bam	8,942,010	0.289168673	8,941,151
ENCFF000XJS_chp2_filt.bam	12,748,871	0.412275775	12,744,729
ENCFF000XKD_chp3_filt.bam	13,217,349	0.427425519	13,212,672

```

chp2=$(samtools view -c ENCF000XJS_chp2_filt.bam)
fact2=$(echo "scale=6; $chp2/$inpc" | bc)
samtools view -b -s $fact2 ENCF000XGP_inp0_filt.bam >
ENCF000XGP_inp0_filt_inp2.bam

chp3=$(samtools view -c ENCF000XKD_chp3_filt.bam)
fact3=$(echo "scale=6; $chp3/$inpc" | bc)
samtools view -b -s $fact3 ENCF000XGP_inp0_filt.bam >
ENCF000XGP_inp0_filt_inp3.bam

```

You can then double check the read count in the new control files for the ChIP-Seq files. The read counts for control files are shown in Table 6.1.

```

samtools view -c ENCF000XGP_inp0_filt_inp1.bam
samtools view -c ENCF000XGP_inp0_filt_inp2.bam
samtools view -c ENCF000XGP_inp0_filt_inp3.bam

```

Up to this point, we have three ChIP-Seq BAM files and three control BAM files, one for each ChIP-Seq file. Before proceeding to the next step, you may decide to view the alignment in the BAM file with the “samtools tview” command or any other BAM viewing program. When we use “samtools tview”, the “-p” option is used to specify a specific position. To view a BAM file, you need to sort the alignments by coordinate and then index it. As an example, we will view only one file (Figure 6.3).

```

samtools sort ENCF000XJP_chp1_filt.bam -o
ENCF000XJP_chp1_filt_so.bam
samtools index ENCF000XJP_chp1_filt_so.bam
samtools tview \

```



FIGURE 6.3 Visualizing reads aligned to the reference genome using “samtools tview” command.

```
ENCFF000XJP_chp1_filt_so.bam \
-p chr17:56084561-56084661 \
../ref/hg19.fa
```

The “samtools tview” provides a quick way to visualize the alignments in a BAM file and to inspect the binding site in a specific location. However, when you need to view a specific gene, you may need to inspect the BAM file to see how the chromosomes are named. You can also find gene coordinates in the reference genome from the NCBI Gene database. Viewing a BAM file is not necessary unless you need to inspect the alignments.

The next step is to use one of the peak-calling programs or shortly a peak caller that uses a ChIP-Seq BAM file and a control BAM file as inputs to perform the process of peak calling as discussed above. There are several peak callers available as open source, but we will use the oldest and the most commonly used one, MACS3.

6.3.4 ChIP-Seq Peak Calling with MACS3

MACS (Model-based Analysis of ChIP-Seq) [6, 7] is one of the popular peak callers, suitable for identifying TF binding sites. MACS utilizes library complexity to evaluate the significance of enriched ChIP peaks by modeling the distribution of reads along the genome using Poisson distribution and then it estimates the FDR for each detected peak empirically. MACS can be used for peak calling using only ChIP-Seq data without providing control or with a control sample (input reads). Sliding window of different sizes is used for finding peaks by calculating the expected number of reads aligned to that region. MACS can perform removal of redundant reads and read shifting. It generates a number of statistics for peak calling.

Basically, MACS separates upstream and downstream aligned reads and aligns them by the midpoint between their peaks. Assume that the distance between the two peaks is d bp, the all reads will be shifted by $d/2$ bp toward 3' ends to cover the most likely DNA protein-binding sites. MACS models read distribution along the genome by Poisson distribution to perform peak calling and to provide p -value. After read shifting, the algorithm uses a sliding window of $2d$ bp to identify candidate peaks with a significant enrichment. It also merges the overlapping peaks and extends reads d bases from their center. The position with the highest fragment pileup is called the summit, which is identified as the exact location of the protein-binding site in the genome. A Poisson distribution parameter λ is estimated for each peak and peaks with p -values less than a threshold will be called. The ratio between the ChIP-Seq read count and λ is estimated as the fold enrichment.

For installation instruction, visit “<https://github.com/macs3-project/MACS>”. On Linux, you can install MACS3 using “`pip install macs3`” and then you can test it by running “`macs3 callpeak -h`” to display the help. In the following, we will use MACS3 to analyze the three ChIP-Seq BAM files while using the corresponding control BAM file. Each “`macs3`” command requires a ChIP-Seq BAM file “`-t`” and a control BAM file “`-c`”. The “`-f`” option specifies the file format of the input file, “`-g`” option specifies the effective genome size, “`-n`” option specifies the string that is prefixed to the output files, “`-q`” option specifies the FDR threshold, “`--bedGraph`” option is to generate a bedGraph file, which is a format that allows

the display of continuous-valued data in a track in genome browsers like UCSC Genome Browser or Integrated Genome Browser (IGB), and “`--outdir`” specifies the directory where the output files are saved.

```
cd ..
mkdir macs3output
macs3 callpeak \
-t bam/ENCF000XJP_chp1_filt.bam \
-c bam/ENCF000XGP_inp0_filt_inp1.bam \
-f BAM \
-g hs \
-n chip1 \
-q 0.05 \
--bdg \
--outdir macs3output
macs3 callpeak \
-t bam/ENCF000XJS_chp2_filt.bam \
-c bam/ENCF000XGP_inp0_filt_inp2.bam \
-f BAM \
-g hs \
-n chip2 \
-q 0.05 \
--bdg \
--outdir macs3output
macs3 callpeak \
-t bam/ENCF000XKD_chp3_filt.bam \
-c bam/ENCF000XGP_inp0_filt_inp3.bam \
-f BAM \
-g hs \
-n chip3 \
-q 0.05 \
--bdg \
--outdir macs3output
```

Several output files for each ChIP-Seq data have been saved in the specified output directory as shown in Figure 6.4. The description of these MACS3 output files is as follows:

Excel file: The “`*_peaks.xls`” is an Excel file, which is the most important file that contains header information in the beginning of the file and information about the peaks called by MACS3 in nine columns. These columns are Chromosome name (chr), start position of peak (start), end position of peak (end), length of peak region (length), absolute peak summit position (abs_summit), pileup height at peak summit (pileup), $-\log_{10}(pvalue)$ for the peak summit ($-\log_{10}(pvalue)$), fold enrichment for this peak summit against random Poisson distribution with local lambda (fold_enrichment), $-\log_{10}(qvalue)$ at peak summit ($-\log_{10}(qvalue)$), and name.

BedGraph format: The “`*_treat_pileup.bdg`” contains the peak enrichment signal and “`*_control_lambda.bdg`” contains local biases for each location in the reference genome.

```

-rw-r--r-- 1 hamiddafa hamiddafa 691M Apr 10 14:06 chip1_control_lambda.bdg
-rw-r--r-- 1 hamiddafa hamiddafa 26K Apr 10 14:46 chip1_model1.pdf
-rw-r--r-- 1 hamiddafa hamiddafa 97K Apr 10 14:05 chip1_model1.r
-rw-r--r-- 1 hamiddafa hamiddafa 2.6M Apr 10 14:06 chip1_peaks.narrowPeak
-rw-r--r-- 1 hamiddafa hamiddafa 2.9M Apr 10 14:06 chip1_peaks.xls
-rw-r--r-- 1 hamiddafa hamiddafa 1.8M Apr 10 14:06 chip1_summits.bed
-rw-r--r-- 1 hamiddafa hamiddafa 402M Apr 10 14:06 chip1_treat_pileup.bdg
-rw-r--r-- 1 hamiddafa hamiddafa 902M Apr 10 14:08 chip2_control_lambda.bdg
-rw-r--r-- 1 hamiddafa hamiddafa 26K Apr 10 14:46 chip2_model1.pdf
-rw-r--r-- 1 hamiddafa hamiddafa 97K Apr 10 14:07 chip2_model1.r
-rw-r--r-- 1 hamiddafa hamiddafa 2.2M Apr 10 14:08 chip2_peaks.narrowPeak
-rw-r--r-- 1 hamiddafa hamiddafa 2.4M Apr 10 14:08 chip2_peaks.xls
-rw-r--r-- 1 hamiddafa hamiddafa 1.5M Apr 10 14:08 chip2_summits.bed
-rw-r--r-- 1 hamiddafa hamiddafa 490M Apr 10 14:08 chip2_treat_pileup.bdg
-rw-r--r-- 1 hamiddafa hamiddafa 920M Apr 10 14:14 chip3_control_lambda.bdg
-rw-r--r-- 1 hamiddafa hamiddafa 26K Apr 10 14:46 chip3_model1.pdf
-rw-r--r-- 1 hamiddafa hamiddafa 97K Apr 10 14:13 chip3_model1.r
-rw-r--r-- 1 hamiddafa hamiddafa 2.6M Apr 10 14:14 chip3_peaks.narrowPeak
-rw-r--r-- 1 hamiddafa hamiddafa 2.9M Apr 10 14:14 chip3_peaks.xls
-rw-r--r-- 1 hamiddafa hamiddafa 1.8M Apr 10 14:14 chip3_summits.bed
-rw-r--r-- 1 hamiddafa hamiddafa 462M Apr 10 14:14 chip3_treat_pileup.bdg

```

FIGURE 6.4 MACS3 output files for the three ChIP-Seq data.

These two files are the largest and they are in bedGraph format that can be visualized in the UCSC browser. The bedGraph format is a format developed to display genomic information in a track on the genomic browser. It consists of four tab-separated columns: chromosome, start, end, and value. The chromosome coordinates are 0-based. The positions (start and end) are listed in ascending order. The data displayed in the track are the values in the value column and they can be integer or real, positive or negative.

BED format: The “*_summits.bed” file is in BED format. The BED (Browser Extensible Data) format defines the data lines that are viewed in an annotation track of a genomic browser. The BED file contains three required fields (chromosome, start, and end) and nine additional optional fields (name, score, strand, thick start, thick end, RGB (an RGB color value), block count, block size, and block start).

The “summits.bed” file contains the summit locations for every peak. The fifth column in this file is the same as what is in the narrowPeak file. This file can be used for finding motifs at the binding sites.

R script file: The “*_model.r” file is an R script to produce a PDF file containing peak model plot and cross-correlation plot. The pdf files for our ChIP-Seq data are generated using the “Rscript” command as follows:

```

Rscript chip1_model.r
Rscript chip2_model.r
Rscript chip3_model.r

```

BED6+4 format: The “*_peaks.narrowPeak” file is in BED6+4 format, which stores information about signal enrichment of the called peaks based on pooled, normalized read counts. This format consists of ten tab-separated columns: chromosome, start, end, name (region name), score (peak density score from 0 to 1000) based on the signal value, strand (+/-), signal value (average enrichment), *p*-value (int(-10*log10pvalue)), FDR or *q*-value

($\text{int}(-10 * \log_{10} \text{qvalue})$, and peak (0-based offset from start). Either p -value or q -value is present in “*_peaks.narrowPeak” file depending on which threshold option is used: “-p” or “-q”.

6.3.5 Visualizing ChIP-Seq Enrichment in Genome Browser

In this step, we will visualize the enrichment signals contained in the bedGraph files (*control_lambda.bdg, and *treat_pileup.bdg) and peak densities in the BED file (*peaks.narrowPeak) in a genome browser like a UCSC Genome Browser or IGB. There are three files for each ChIP-Seq sample. First, we need to convert bedGraph format into BigWig format, which is an indexed binary file created from bedGraph file using “bedGraphToBigWig” utility for fast display. We can download the conversion utilities from the UCSC using the following commands (download these programs in the directory where the MACS3 output files are found for easy use):

```
rsync -aP \
    rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.x86_64/
bedGraphToBigWig ./
rsync -aP \
    rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.x86_64/
bigWigToWig ./
rsync -aP \
    rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.x86_64/
fetchChromSizes ./
```

The “bedGraphToBigWig” utility converts a file from the bedGraph format to the BigWig format. The “bigWigToWig” utility converts a file from the BigWig format to the Wig format. The “fetchChromSizes” utility creates a text file (from a reference genome) containing the chromosome names and sizes in bases. For more information about these commands and file format, refer to the UCSC website.

Before proceeding, we need to create a text file containing the name of the human chromosomes and their sizes using “fetchChromSizes”.

```
fetchChromSizes hg19 > hg19.chrom.sizes
```

The file “hg19.chrom.sizes” will be created in the working directory; you can view it using “less hg19.chrom.sizes”. This file consists of two columns: chromosome name and chromosome size in bases.

Now, we can convert the bedGraph files to BigWig format and then from BigWig format to Wig format using the following commands while the working directory is the one where the MACS3 output files are found:

```
mkdir vis
#Convert *control_lambda.bdg from bedGraph to BigWig
bedGraphToBigWig \
```

```

chip1_control_lambda.bdg hg19.chrom.sizes \
vis/chip1_control_lambda.bw
bedGraphToBigWig \
  chip2_control_lambda.bdg hg19.chrom.sizes \
  vis/chip2_control_lambda.bw
bedGraphToBigWig \
  chip3_control_lambda.bdg hg19.chrom.sizes \
  vis/chip3_control_lambda.bw

#Convert *treat_pileup.bdg from bedGraph to BigWig
bedGraphToBigWig \
  chip1_treat_pileup.bdg \
  hg19.chrom.sizes \
  vis/chip1_treat_pileup.bw
bedGraphToBigWig \
  chip2_treat_pileup.bdg \
  hg19.chrom.sizes \
  vis/chip2_treat_pileup.bw
bedGraphToBigWig \
  chip3_treat_pileup.bdg \
  hg19.chrom.sizes \
  vis/chip3_treat_pileup.bw

#Convert control_lambda.bw from Bigwig to wig
bigWigToWig \
  vis/chip1_control_lambda.bw \
  vis/chip1_control_lambda.wig
bigWigToWig \
  vis/chip2_control_lambda.bw \
  vis/chip2_control_lambda.wig
bigWigToWig \
  vis/chip3_control_lambda.bw \
  vis/chip3_control_lambda.wig

#Convert chip1_treat_pileup.bw from Bigwig to wig
bigWigToWig \
  vis/chip1_treat_pileup.bw \
  vis/chip1_treat_pileup.wig
bigWigToWig \
  vis/chip2_treat_pileup.bw \
  vis/chip2_treat_pileup.wig
bigWigToWig \
  vis/chip3_treat_pileup.bw \
  vis/chip3_treat_pileup.wig

```

We need also to modify the BED file “peaks.narrowPeak” by keeping only columns 1–4.

```
cut -f1,2,3,4 chip1_peaks.narrowPeak > vis/chip1_peaks.bed
cut -f1,2,3,4 chip2_peaks.narrowPeak > vis/chip2_peaks.bed
cut -f1,2,3,4 chip3_peaks.narrowPeak > vis/chip3_peaks.bed
```

The converted files were saved in “macs3output/vis” directory as shown in Figure 6.5. Those files can be visualized in a genome browser. If you are working with a Linux with a graphical desktop, that works for the next step; otherwise, you can copy “macs3output” directory including “vis” to a Windows or Mac desktop using FileZilla, which is an open-source FTP application for file transfer, available at “<https://filezilla-project.org/>”. The “macs3output” directory contains all MACS3 output files that we will use in later analyses and “vis” sub-directory contains files for visualization in a genome browser.

In the next step, we will use a genome browser. In this exercise, we will use the IGB [8], which is a standalone software available for Linux, Windows, and Mac OS X. It can be downloaded from “<https://www.bioviz.org/>” and installed on a local computer. Once it has been installed, open it, click “H. sapiens”, open the directory where the Wig and BED files are stored, and for each ChIP-Seq sample, use the mouse to drag “*control_lambda.wig”, “*treat_pileup.wig”, and “*peaks.bed” to the viewer just above “RefSeq curated (+)” track and every time click “Load Data” button on the top right. You can also change the color of the tracks of each sample by clicking the right button on the track and select “Customize” from the popup menu and then choose a color (Figure 6.6).

As shown in Figure 6.6, each ChIP-Seq sample has three tracks: control (input data), treated (ChIP-Seq), and the peaks. It is clear that the signal of input data (control) is flat but the ChIP-Seq signals are with peaks. The peak track shows the peak position. Compare the control signal of each sample to its corresponding treated (ChIP-Seq) signal. You can move along the genome by using grab tool (hand icon), or zoom in and out using the horizontal zoom slider on the top. You can also use the mouse to move left or right. The IGB comes with search tools that can be used to search for a gene or to navigate to a specific position in a chromosome.

Figure 6.7 shows a typical mixed (sharp and broad) signal of Poly II for the three samples after zooming in.

```
total 4.2G
-rw-r--r-- 1 hamiddafa hamiddafa 119M Apr 10 15:56 chip1_control_lambdabw
-rw-r--r-- 1 hamiddafa hamiddafa 692M Apr 10 15:58 chip1_control_lambda.wig
-rw-r--r-- 1 hamiddafa hamiddafa 1.5M Apr 10 16:24 chip1_peaks.bed
-rw-r--r-- 1 hamiddafa hamiddafa 59M Apr 10 15:57 chip1_treat_pileupbw
-rw-r--r-- 1 hamiddafa hamiddafa 327M Apr 10 15:59 chip1_treat_pileup.wig
-rw-r--r-- 1 hamiddafa hamiddafa 155M Apr 10 15:56 chip2_control_lambdabw
-rw-r--r-- 1 hamiddafa hamiddafa 884M Apr 10 15:59 chip2_control_lambda.wig
-rw-r--r-- 1 hamiddafa hamiddafa 1.3M Apr 10 16:24 chip2_peaks.bed
-rw-r--r-- 1 hamiddafa hamiddafa 72M Apr 10 15:58 chip2_treat_pileupbw
-rw-r--r-- 1 hamiddafa hamiddafa 399M Apr 10 15:59 chip2_treat_pileup.wig
-rw-r--r-- 1 hamiddafa hamiddafa 157M Apr 10 15:57 chip3_control_lambdabw
-rw-r--r-- 1 hamiddafa hamiddafa 908M Apr 10 15:59 chip3_control_lambda.wig
-rw-r--r-- 1 hamiddafa hamiddafa 1.3M Apr 10 16:24 chip3_peaks.bed
-rw-r--r-- 1 hamiddafa hamiddafa 66M Apr 10 15:58 chip3_treat_pileupbw
-rw-r--r-- 1 hamiddafa hamiddafa 377M Apr 10 15:59 chip3_treat_pileup.wig
```

FIGURE 6.5 Wig and BED files ready to be visualized in a genome browser.

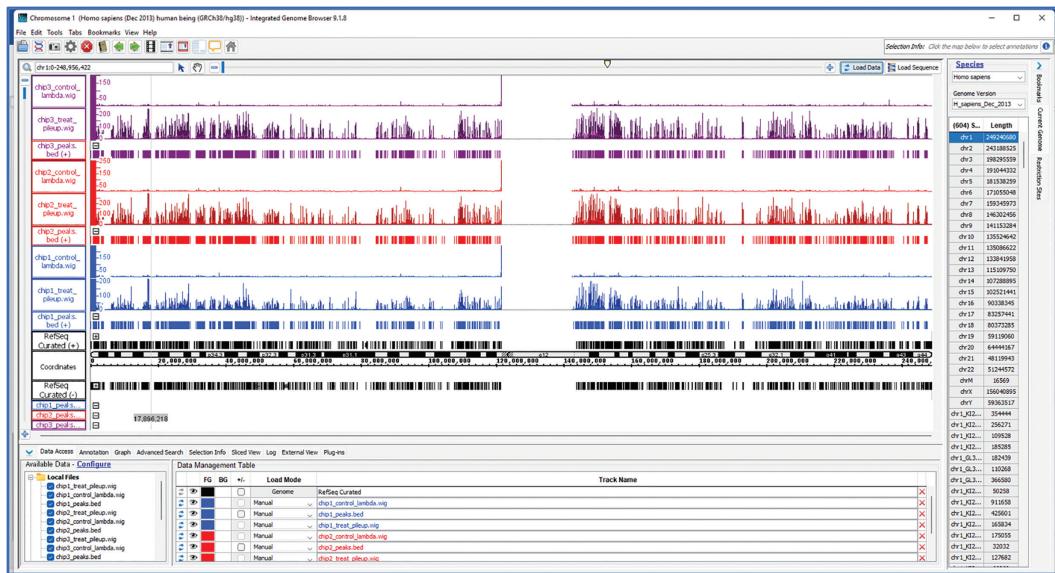


FIGURE 6.6 The IGB displaying the tracks of the three ChIP-Seq samples.

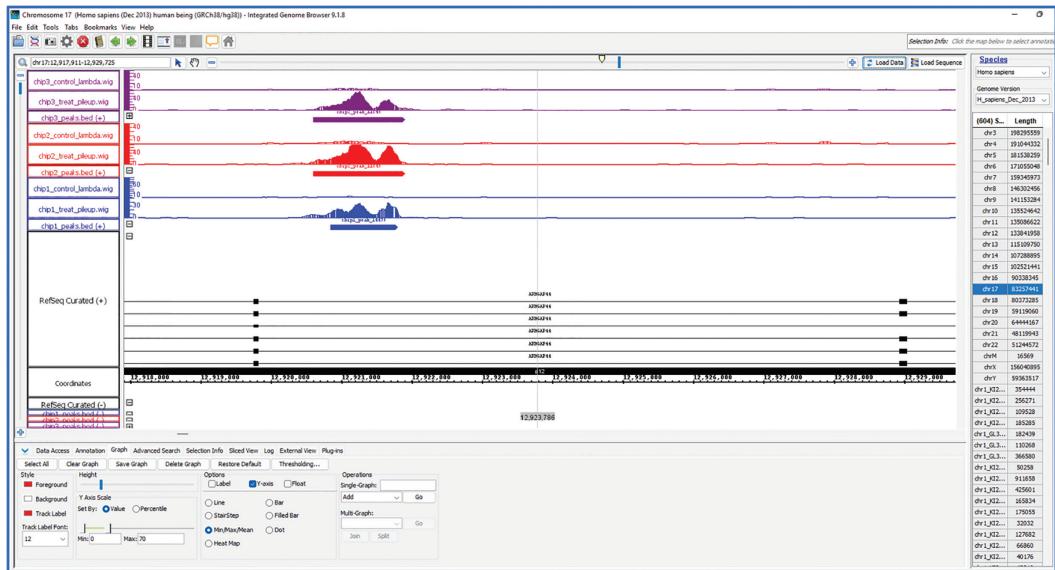


FIGURE 6.7 The IGB displaying POLR2A TF binding site.

6.3.6 Visualizing Peaks Distribution

In this time, we will use peaks files “*peaks.narrowPeak” to visualize the peak enrichment distribution using R Bioconductor packages. You can use R in any platform (Linux, Windows, or Mac OS). Follow the instructions at “<https://cran.r-project.org/>” to download

and install R. You can also download and install RStudio by following the instructions at “<https://www.rstudio.com/products/rstudio/download/>”. Once you have R installed on your computer, run R, and on R prompt, run the following to install the Bioconductor packages required for the remaining ChIP-Seq data analysis:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("clusterProfiler")
BiocManager::install("ChIPseeker")
BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
BiocManager::install("EnsDb.Hsapiens.v75")
BiocManager::install("clusterProfiler")
BiocManager::install("AnnotationDbi")
BiocManager::install("org.Hs.eg.db")
install.packages("dplyr")
```

After installing the packages, load them as follows:

```
library(clusterProfiler)
library(ChIPseeker)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(EnsDb.Hsapiens.v75)
library(AnnotationDbi)
library(org.Hs.eg.db)
library("dplyr")
```

Now, we are ready to finish the remaining analysis in R. Copy “chip1_peaks.narrowPeak”, “chip2_peaks.narrowPeak”, and “chip3_peaks.narrowPeak” files produced by MACS3 above to a directory that you can browse from inside R. Use R to choose your working directory where those three files are copied. In the following, we can use ChIPseeker[9] package functions to create different plots.

6.3.6.1 ChIP-Seq Peaks’ Coverage Plot

The “covplot()” function is used to create a plot that shows peak distribution over the whole genome. This function calculates the coverage of peak regions over chromosomes or regions of chromosomes and generates a peaks’ coverage plot. The function requires the peak data as Granges (genomic ranges) object (see ChIPseeker documentations). The following codes load each file as a data frame, add column names, create a Granges object, and finally create the ChIP-Seq peaks’ coverage plots for each sample:

```
peaks1 <- read.table("chip1_peaks.narrowPeak", header=FALSE)
colnames <- c("chrom", "start", "end", "name", "score", "strand",
"signal", "pvalue", "qvalue", "peak")
colnames(peaks1) <- colnames
```

```

peaks2<- read.table("chip2_peaks.narrowPeak", header=FALSE)
colnames(peaks2) <- colnames
colnames(peaks2)
peaks3<- read.table("chip3_peaks.narrowPeak", header=FALSE)
colnames(peaks3) <- colnames

#head(peaks1)
peaks1Ranges<- GRanges(seqnames=peaks1$chrom,
                        ranges=IRanges(peaks1$start,peaks1$end),
                        peaks1$name,
                        peaks1$score,
                        strand=NULL,
                        peaks1$signal,
                        peaks1$pvalue,
                        peaks1$qvalue,
                        peaks1$peak)
covplot(peaks1Ranges, weightCol="peaks1$peak")

peaks2Ranges<- GRanges(seqnames=peaks2$chrom,
                        ranges=IRanges(peaks2$start,peaks2$end),
                        peaks2$name,
                        peaks2$score,
                        strand=NULL,
                        peaks2$signal,
                        peaks2$pvalue,
                        peaks2$qvalue,
                        peaks2$peak)
covplot(peaks2Ranges, weightCol="peaks2$peak")

peaks3Ranges<- GRanges(seqnames=peaks3$chrom,
                        ranges=IRanges(peaks3$start,peaks3$end),
                        peaks3$name,
                        peaks3$score,
                        strand=NULL,
                        peaks3$signal,
                        peaks3$pvalue,
                        peaks3$qvalue,
                        peaks3$peak)
covplot(peaks3Ranges, weightCol="peaks3$peak")

```

Three ChIP-Seq peak coverage plots will be created but we will display only a single plot to save space. Figure 6.8 shows the coverage plot for the first sample (chip1); it shows the distribution of the peaks of each human chromosome.

Rather than the entire genome, “covplot()” can also display the coverage in a single chromosome, a group of chromosome, a specific region of a chromosome, or it can be

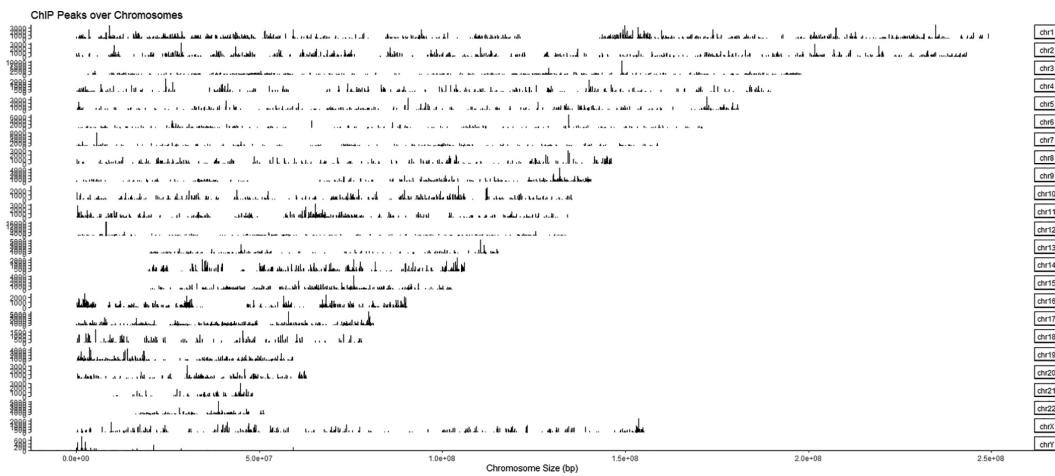


FIGURE 6.8 ChIP-Seq peak coverage plot.

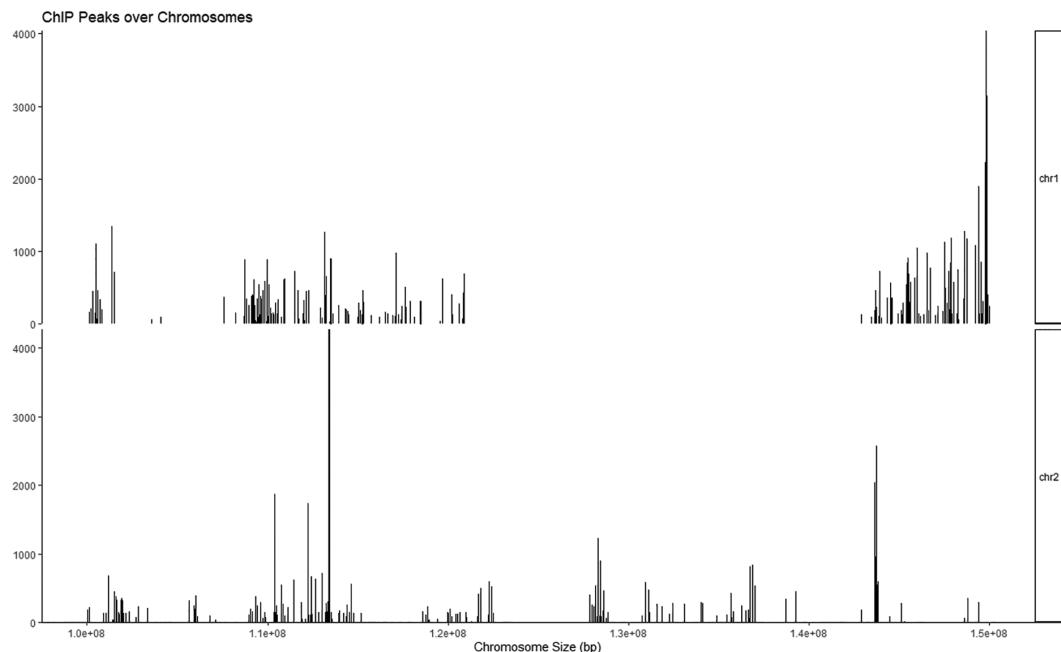


FIGURE 6.9 ChIP-Seq peak coverage plot comparing between a region in Chromosomes 1 and 2.

used to compare between chromosomes. The following codes display peak coverage in the region between $1.0\text{e}8$ and $1.5\text{e}8$ bp for chromosome 1 and 2 (Figure 6.9):

```
covplot(peaks1Ranges, weightCol="peaks1$peak",
        chrs=c("chr1", "chr2"), xlim=c(1.0e8, 1.5e8))
```

6.3.6.2 Distribution of Peaks in Transcription Start Site (TSS) Regions

The heatmap can be used to show the peak distribution on the transcription start site (TSS) region as shown in Figure 6.10.

```
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
promoter <- getPromoters(TxDb=txdb, upstream=2000,
downstream=2000)
tagMatrix <- getTagMatrix(peaks1Ranges, windows=promoter)
tagHeatmap(tagMatrix, xlim=c(-2000, 2000), color="blue")
```

The distribution of peaks in the TSS region can also be visualized with the line plot that profiles the average peaks in the TSS region.

```
plotAvgProf(tagMatrix, xlim=c(-2000, 2000),
xlab="Genomic Region (5' ->3')",
ylab = "Read Count Frequency")
```

We can notice that as shown in Figure 6.11, the peaks are normally distributed (bell-shaped) with the mean in the TSS region.

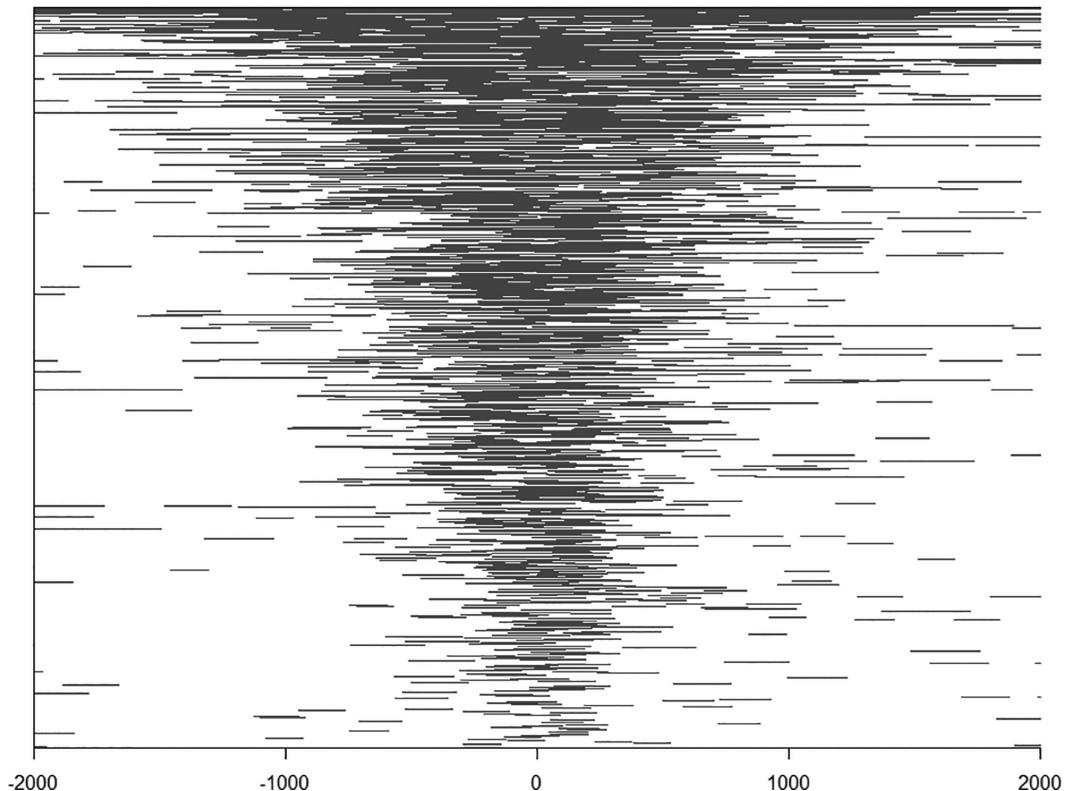


FIGURE 6.10 ChIP-Seq peak profiling in the TSS regions of the genes.

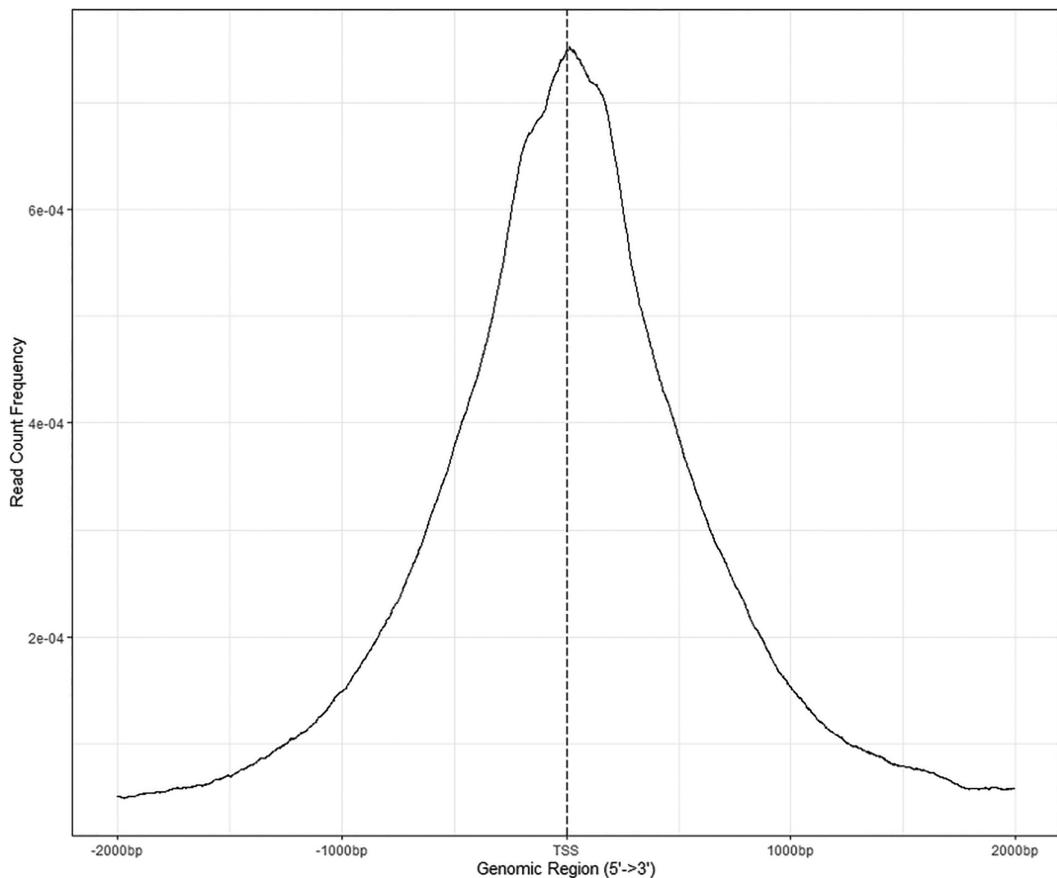


FIGURE 6.11 Average profile of ChIP-Seq peaks.

A line plot provided with confidence interval can also be created. The confidence interval is estimated by resampling the peaks several times without replacement and computing the average and variance each time.

```
plotAvgProf(tagMatrix, xlim=c(-2000, 2000),
            conf = 0.95, resample = 1000)
```

6.3.6.3 Profile of Peaks along Gene Regions

The average peak profile in the different gene regions across genes will give a clear idea about the site of the protein–DNA interaction. The “plotPeakProf2()” creates a profile plot for the peak signal from the TSSs to the transcription termination sites (TTSSs).

```
#Profile of ChIP peaks binding to body regions
plotPeakProf2(peak = peaks1Ranges, upstream = rel(0.2), downstream = rel(0.2),
              conf = 0.95, by = "gene", type = "body", nbin = 800, TxDb = txdb,
              weightCol = "peaks1$peak", ignore_strand = F)
```

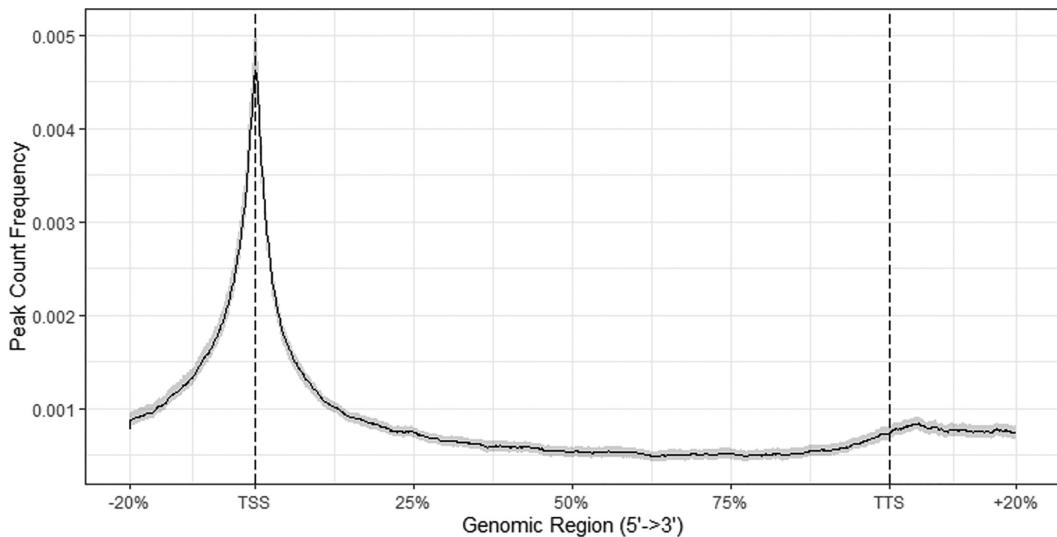


FIGURE 6.12 Average profile of ChIP-Seq peaks across.

Figure 6.12 shows that Poly II localization is centered in the TSS where most peaks are observed.

6.3.7 Peak Annotation

We will continue using R to perform annotation of the peaks called with MACS3 program that stored the peak information in “*peaks.narrowPeak” files. The peaks represent the most likely locations of protein–DNA interaction in the genome (the content of “*peaks.narrowPeak” files is discussed above). The main goal of ChIP-Seq data analysis is to investigate the biological implications of the epigenomic changes like genomic binding sites of proteins such as TFs, histones, and Poly II. Annotation of the protein–DNA interaction sites and functions will provide important information about the biological implications. Peak annotation is the process of associating the sites identified by the peaks to the genes and region of the genes affected by the epigenetic change. Most interactions like TFs, initial localization of Poly II and histones occur in the cis-regulatory site of the gene which is close to TSS and it includes a promoter, an enhancer, a silencer, insulators, etc., which play crucial roles in controlling gene expressions in specific cell types, conditions, and developmental stages. An annotation program annotates ChIP-Seq peaks by associating these peaks to the closest TSS of a gene, either upstream or downstream. A cis-regulatory region can also be in distance from the TSS or between the TSSs of two different genes.

We will continue using R Bioconductor packages and “*peaks.narrowPeak” files as inputs for annotation. The following codes create a list of the sample file names and a label for each sample as “chip1”, “chip2”, and “chip3” respectively:

```
bedfiles <- list.files("vis", pattern= ".bed", full.names=T)
bedfiles <- as.list(bedfiles)
names(bedfiles) <- c("chip1", "chip2", "chip3")
```

Then, we can assign the database of the known human genes to a variable so that we can use the annotation information and associate them to the peaks.

```
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```

The above annotation data package is for human (*Homo sapiens*) data from UCSC build hg19 based on the knownGene Track. The available annotation packages for the genomes of other organisms are available at “<http://bioconductor.org/packages/3.5/data/annotation/>”.

In the next step, we will use “`annotatePeak()`” function from ChIPseeker package[9] to annotate the peaks by associating them to the nearest genes. This function also provides the option “`tssRegion=`” that allows us to specify a max distance from the TSS in which the peaks can be associated to the gene.

```
annotated_peaks <- lapply(bedfiles,
                           annotatePeak,
                           TxDb=txdb,
                           tssRegion=c(-1000, 1000), verbose=FALSE)
annotated_peaks
```

The above R codes apply the “`annotatePeak()`” function to annotate the peaks in the peak signal files. The peak region was also set to any distance in the range (-1000, 1000) from the TSS of the gene. Figure 6.13 shows the annotation summaries for the three samples. The summary includes the number of peaks annotated on the top and then the peak annotation frequencies based on the genomic features (gene regions). We can notice that the maximum frequencies are in the promoter region, which in this case is an indication for the transcriptional activity in the gene associated to the peaks.

The ChIPseeker package provides several functions to visualize the annotated peaks. The “`plotAnnoBar()`” creates a bar chart for the peak representation in the different genomic regions (features).

```
plotAnnoBar(annotated_peaks)
```

Figure 6.14 shows a bar plot that depicts peak enrichment representation in the different genomic regions of the genes. We can notice that most peaks are centered in the promoter regions. This may look different if the ChIP-Seq is for TFs or histone marks.

Distribution of peaks relative to TSS:

The sites of TF binding and Poly II localization are found in the promoter regions of the genes. Thus, distribution of peaks around TSS will give an idea about the activity of the

\$chip1	\$chip2	\$chip3
Annotated peaks generated by ChIPseeker 36275/36275 peaks were annotated	Annotated peaks generated by ChIPseeker 30160/30160 peaks were annotated	Annotated peaks generated by ChIPseeker 30160/30160 peaks were annotated
Genomic Annotation Summary:	Genomic Annotation Summary:	Genomic Annotation Summary:
Feature Frequency	Feature Frequency	Feature Frequency
9 Promoter 46.1226740	9 Promoter 52.0125995	9 Promoter 52.0125995
4 5' UTR 0.4962095	4 5' UTR 0.5968170	4 5' UTR 0.5968170
3 3' UTR 5.0778773	3 3' UTR 4.6485411	3 3' UTR 4.6485411
1 1st Exon 0.6753963	1 1st Exon 0.7990716	1 1st Exon 0.7990716
7 Other Exon 3.8042729	7 Other Exon 4.0716180	7 Other Exon 4.0716180
2 1st Intron 9.8166782	2 1st Intron 7.3275862	2 1st Intron 7.3275862
8 Other Intron 13.9545141	8 Other Intron 10.4177719	8 Other Intron 10.4177719
6 Downstream (<=300) 0.4658856	6 Downstream (<=300) 0.4641910	6 Downstream (<=300) 0.4641910
5 Distal Intergenic 19.5864921	5 Distal Intergenic 19.6618037	5 Distal Intergenic 19.6618037

FIGURE 6.13 Annotation summaries for three ChIP-Seq samples.

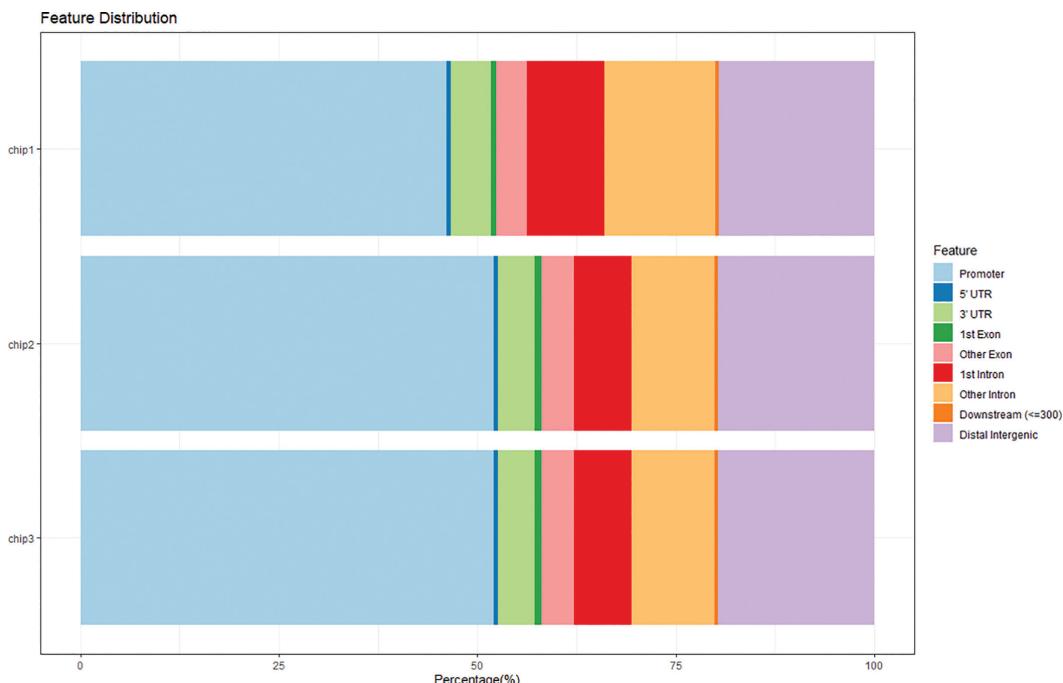


FIGURE 6.14 Bar chart of the genomic feature representation.

protein studied. The “plotDistToTSS()” function creates a plot showing the distribution of the peaks relative to the TSS.

```
plotDistToTSS(annotated_peaks,
              title="Distribution of Poly II relative to TSS")
```

As shown in Figure 6.15, most interaction sites are in the region of 0–1 kb from the TSS of the genes.

6.3.7.1 Writing Annotations to Files

The annotation information of the peaks can be written to a file so that it can be used for further analysis. The annotation file will include chromosome name, start (peak start position), end (peak end position), width (number of bases), strand, peak name, annotation (gene region), gene chromosome, gene start, gene end, gene length (bases), gene strand, gene Id, transcript Id, distance to TSS (in bases), and gene name. This information is usually used by researchers to investigate individual genes. The following R codes write the annotations for the three ChIP-Seq samples to three text files: “Chip1_peak_annotation.txt”, “Chip2_peak_annotation.txt”, and “Chip3_peak_annotation.txt”.

```
#Write Chip1 annotation to a file
#separate Chip1 annotation in a data frame
chip1_annot <- data.frame(annotated_peaks[["chip1"]])@anno
```

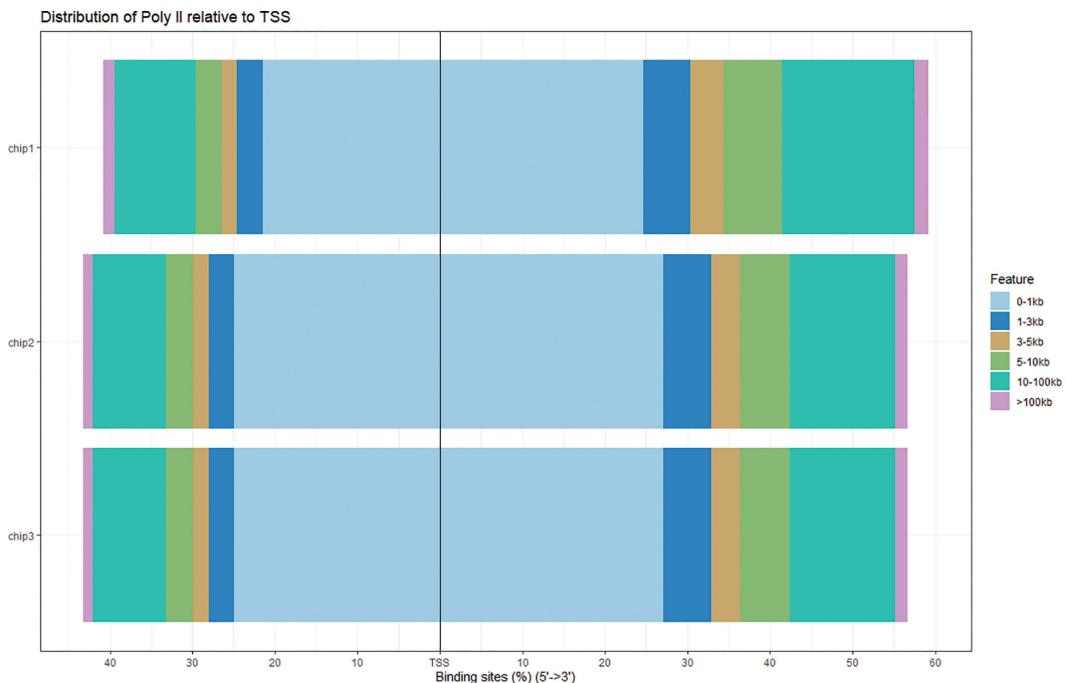


FIGURE 6.15 Distribution of Poly II relative to the TSS.

```

annotations_edb2$ENTREZID
<- as.character(annotations_edb2$ENTREZID)
chip2_annot %>% left_join(annotations_edb2,
                           by=c("geneId"="ENTREZID")) %>%
write.table(file="Chip2_peak_annotation.txt",
            sep="\t", quote=F, row.names=F)

# Write Chip3 annotation to a file
chip3_annot <- data.frame(annotated_peaks[["chip3"]][@anno])
entrez3 <- chip3_annot$geneId
annotations_edb3 <- AnnotationDbi::select(EnsDb.Hsapiens.v75,
                                           keys = entrez3,
                                           columns = c("GENENAME"),
                                           keytype = "ENTREZID")

annotations_edb3$ENTREZID
<- as.character(annotations_edb3$ENTREZID)
chip3_annot %>% left_join(annotations_edb3,
                           by=c("geneId"="ENTREZID")) %>%
write.table(file="Chip3_peak_annotation.txt",
            sep="\t", quote=F, row.names=F)

```

Those files will be saved in the working directory. Open these files in Excel to study their contents.

6.3.8 ChIP-Seq Functional Analysis

After peak annotation and functional enrichment, the next step in ChIP-Seq data analysis is the identification of the biological implications of the expression of the genes associated with the sites. For this purpose, we will use knowledge from Gene Ontology (GO) and KEGG pathway and other pathway databases as we did in the RNA-Seq data analysis. The GO enrichment analysis is performed using “enrichGO()”, which is one of the “clusterProfiler” package [10]. This function will return the enrichment GO categories based on the FDR threshold. The following codes perform GO analysis for each sample and then write the results into a file and create a dot plot showing a specified number of the top GO terms. The dot size on the plot represents the number of the genes related to GO term divided by the total number of significant genes and the size of the circle describes the significance in adjusted *p*-value (p-adjusted).

```

ego1 <- enrichGO(gene = entrez1,
                  keyType = "ENTREZID",
                  OrgDb = org.Hs.eg.db,
                  ont = "BP",
                  pAdjustMethod = "BH",
                  qvalueCutoff = 0.05,
                  readable = TRUE)
ego2 <- enrichGO(gene = entrez2,

```

```

keyType = "ENTREZID",
OrgDb = org.Hs.eg.db,
ont = "BP",
pAdjustMethod = "BH",
qvalueCutoff = 0.05,
readable = TRUE)

ego3 <- enrichGO(gene = entrez3,
                  keyType = "ENTREZID",
                  OrgDb = org.Hs.eg.db,
                  ont = "BP",
                  pAdjustMethod = "BH",
                  qvalueCutoff = 0.05,
                  readable = TRUE)

#GO output
# Chip1
cluster_summary1 <- data.frame(ego1)
write.csv(cluster_summary1, "chip1_GO.csv")
# Dotplot visualization
dotplot(ego1, showCategory=10)
# Chip2
cluster_summary2 <- data.frame(ego2)
write.csv(cluster_summary2, "chip2_GO.csv")
# Dotplot visualization
dotplot(ego2, showCategory=10)
# Chip2
cluster_summary3 <- data.frame(ego3)
write.csv(cluster_summary3, "chip3_GO.csv")
# Dotplot visualization
dotplot(ego3, showCategory=10)

```

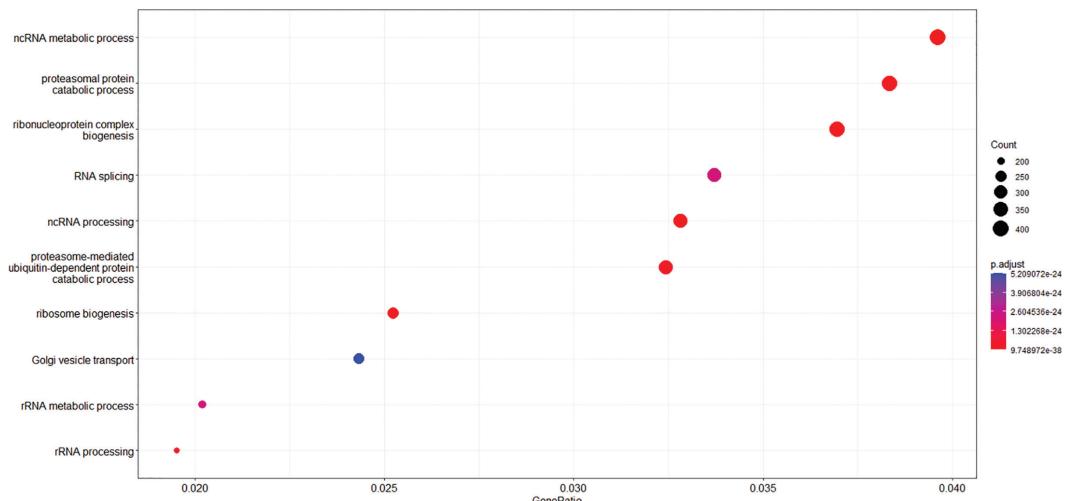


FIGURE 6.16 GO dot plot showing the top 10 genes.

Figure 6.16 shows the dot plot of the first ChIP-Seq sample.

The IDs, descriptions, and statistics of the significant GO terms are stored in “chip1_GO.csv”, “chip2_GO.csv”, and “chip3_GO.csv”. In Figure 6.16, we can notice that those top ten GO terms are associated with gene transcription which reflects the Poly II biological activity. The definitions of the GO terms can be searched at “http://www.informatics.jax.org/vocab/gene_ontology/”. Thus, ChIP-Seq provides information about the functions of the protein studied.

We can also use KEGG database for gene pathways to annotate the genes with significant peaks. The “enrichKEGG()” function returns the enrichment KEGG categories with FDR control. The following codes generate KEGG signaling pathway annotation and create dot plot for each sample (Figure 6.17):

```
ekegg1 <- enrichKEGG(gene = entrez1, organism = 'hsa',
pvalueCutoff = 0.05)
cluster_kegg1 <- data.frame(ekegg1)
write.csv(cluster_kegg1, "kegg_chip1.csv")
dotplot(ekegg1)
#Chip2
ekegg2 <- enrichKEGG(gene = entrez2, organism = 'hsa',
pvalueCutoff = 0.05)
cluster_kegg2 <- data.frame(ekegg2)
write.csv(cluster_kegg2, "kegg_chip2.csv")
dotplot(ekegg2)
#Chip3
ekegg3 <- enrichKEGG(gene = entrez3, organism = 'hsa',
pvalueCutoff = 0.05)
cluster_kegg3 <- data.frame(ekegg3)
write.csv(cluster_kegg3, "kegg_chip3.csv")
dotplot(ekegg3)
```

The significant KEGG signaling pathways show the most likely active pathways in the cells.

We can also compare enrichment across samples by using “compareCluste()” function, which requires the list of genes from each sample (Figure 6.18).

```
# Create a list with genes from each sample
genes = lapply(annotated_peaks, function(i) as.data.
frame(i)$geneId)

# Run KEGG analysis
compKEGG <- compareCluster(geneCluster = genes,
                             fun = "enrichKEGG",
                             organism = "human",
                             pvalueCutoff = 0.05,
                             pAdjustMethod = "BH")
dotplot(compKEGG, showCategory = 10, title = "KEGG Pathway
Enrichment Analysis")
```

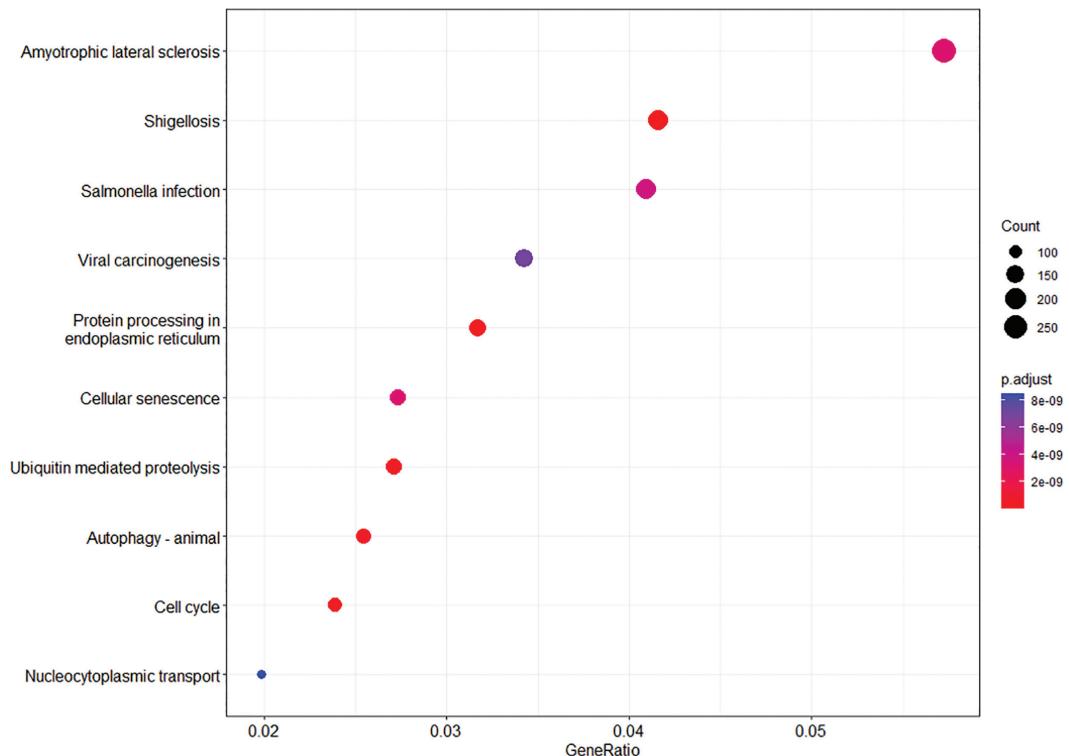


FIGURE 6.17 KEGG dot plot showing the top 10 genes.

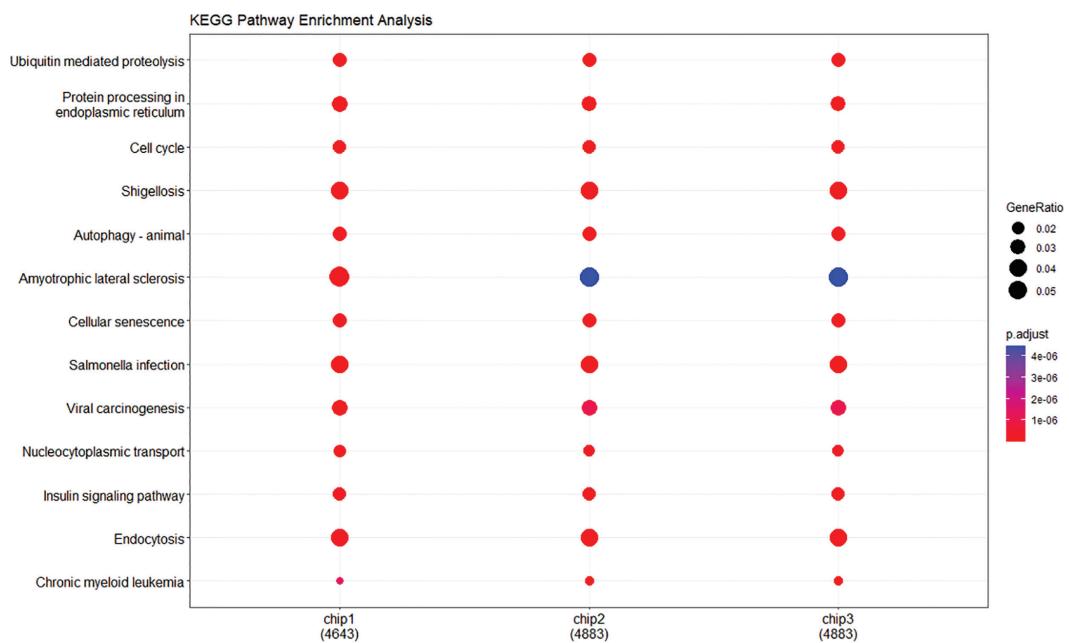


FIGURE 6.18 KEGG signaling pathway comparison across the three samples.

6.3.9 Motif Discovery

The major goal of ChIP-Seq is the determination of the binding sites, where TFs, Poly II, and histone marks interact with the genomic DNA to control the transcription of genes. Those sites have sequence patterns that are recognized by the targeted proteins. The genomic sequence pattern that has such biological activity is called a motif. The motifs are usually found in the genes' regulatory regions. Therefore, they are most likely to be found in the peak enrichment regions. The motif enrichment analysis is used to detect enrichment of known binding motifs in the regulatory regions of genes. The researchers use motif analysis to detect the binding site patterns of known library of TFs that are believed to regulate a specific set of genes. Motifs are searched around the ChIP-Seq peaks of a specified window size. Remember that we have peak enrichment stored in “*peaks.narrowPeak” files. However, the motif detection programs require FASTA sequence as input. Therefore, we need to generate FASTA sequences from the BED file. We can create BED files by extracting the first three columns from “*peaks.narrowPeak” files as follows:

```
mkdir motifs
cut -f 1,2,3 \
  macs3output/chip1_peaks.narrowPeak \
  > motifs/chip1_peaks.bed
cut -f 1,2,3 \
  macs3output/chip2_peaks.narrowPeak \
  > motifs/chip2_peaks.bed
cut -f 1,2,3 \
  macs3output/chip3_peaks.narrowPeak \
  > motifs/chip3_peaks.bed
```

The above commands create a new directory, “motifs”, and store the new created BED files in it. We will extract FASTA sequences from each of these three files using bedtools, which is a collection of programs for manipulation of BED files. On Ubuntu, you can install bedtools using “apt-get install bedtools”.

Visit the program website “<https://bedtools.readthedocs.io/en/latest/content/installation.html>” for more information.

The “bedtools getfasta” command is used to extract a FASTA file from each BED file. This command requires the FASTA file of the reference sequence and a bed file as input. We will use the same reference sequence that we used to generate BAM files.

```
bedtools getfasta \
  -fi ref/hg19.fa \
  -bed motifs/chip1_peaks.bed \
  -fo motifs/chip1_peaks.fasta
bedtools getfasta \
  -fi ref/hg19.fa \
  -bed motifs/chip2_peaks.bed \
  -fo motifs/chip2_peaks.fasta
```

```
bedtools getfasta \
-fi ref/hg19.fa \
-bed motifs/chip3_peaks.bed \
-fo motifs/chip3_peaks.fasta
```

Those three FASTA files contain the sequences of enriched peaks for each sample and we will use them as inputs for the motif detection programs.

There are two approaches for motif detection: de novo method when no prior information is assumed and a position weight matrix (PWM) method for known motif.

The de novo approach searches for motifs in an input FASTA sequences without prior information about the motifs. The search is conducted in a window around the peak. The motif discovery programs either create k-mers from the sequences and perform exhaustive search to identify the most frequent consensus substring of the sequences as motifs or use sequence alignments iteratively to create consensus motifs from the PWM that identifies motifs as the consensus motifs with the most frequent nucleobases. An example of de novo motif discovery program is MEME Suite [11], which has DREME, MEME, or STREME programs for discovering ungapped motifs. DREME is k-mer based, but it is deprecated and will not be supported in the future. MEME is an alignment-based motif discovery tool but it is recommended for motifs discovery in less than 50 sequences. STREME is a k-mer based and it is recommended for detecting motifs in a dataset with more than 50 sequences. MEME SUITE is available as web server and command-line programs. To use the web server or to download and install MEME SUITE, visit "<https://meme-suite.org/meme/>". On Linux, you can download and install MEME SUITE by using the following steps:

```
wget https://meme-suite.org/meme/meme-software/5.4.1/meme-5.4.1.tar.gz
tar vxf meme-5.4.1.tar.gz
cd meme-5.4.1
./configure --prefix=$HOME/meme --enable-build-libxml2
--enable-build-libxslt
make
make test
make install
```

Once you have installed it, you can add the following to ".bashrc" file:

```
export PATH=$HOME/meme/bin:$HOME/meme/libexec/meme-5.4.1:$PATH
```

The version may change so the best way is to visit the MEME SUITE website for the latest installation instruction.

After adding the above line to the ".bashrc" file, you may need to restart the terminal or use "source ~/.bashrc" for the change that you have made to take effect.

The MEME Suite programs require the ChIP-Seq dataset in FASTA (primary dataset) and control dataset (secondary dataset). If no control dataset is used, MEME Suite

programs will create a control dataset by shuffling each of the sequences in the primary input dataset.

The following DREME command will search for motif in the FASTA sequences of the three ChIP-Seq samples. However, because the process may take a long time and this is just a practice, you can run the command for a single sample only to save time. Run the commands from inside “motifs” directories, where FASTA files are found.

```
dreme -verbosity 2 \
    -oc dreme_motifs_chip1 \
    -dna \
    -p chip1_peaks.fasta \
    -t 14400 \
    -e 0.05
dreme -verbosity 2 \
    -oc dreme_motifs_chip2 \
    -dna \
    -p chip2_peaks.fasta \
    -t 14400 \
    -e 0.05
dreme -verbosity 2 \
    -oc dreme_motifs_chip3 \
    -dna \
    -p chip3_peaks.fasta \
    -t 14400 \
    -e 0.05
```

The “-oc” specifies the output directory, “-dna” specifies the type of sequence, “-p” specifies the primary dataset, “-t” specifies an elapsed time as a stopping criterion, and “-e” specifies the E-value threshold.

The output files will be saved in the directories “dreme_motifs_chip”. The motifs are reported in an HTML file, an XML file, and a text file. You can open each of these files by using the right program. You can change into each of the output directory and display the HTML file using Firefox as follows:

```
firefox dreme.html
```

Figure 6.19 shows the motifs as displayed on the HTML file. The figure shows motif sequence, logo, RC logo (reverse complement logo), and E-value. The motif sequence logo is a graphical representation of the sequence conservation of DNA nucleotides. A DNA sequence logo consists of the four nucleobase letters A, C, G, and T at each position. The relative sizes of the letters reflect their frequency in the aligned sequences. The sequence of the motif uses the IUPAC codes for nucleotides for representing each of the 15 possible combinations as shown in Table 6.2.

Motif	Logo	RC Logo	E-value	Unerased E-value	More	Submit/Download
1. CHCCWCCC			2.5e-466	3.5e-284	↓	...
2. GCBGCCGC			4.9e-371	2.7e-230	↓	...
3. VAGGAAR			1.2e-357	4.3e-365	↓	...
4. CCCHGCCC			1.3e-311	8.4e-262	↓	...
5. AAANAAA			1.3e-330	1.8e-216	↓	...
6. AHAYACA			3.8e-257	9.8e-277	↓	...
7. DAAATR			2.6e-230	1.1e-315	↓	...
8. CMKCCTC			1.6e-221	1.0e-189	↓	...
9. ACGTGD			1.3e-156	9.8e-175	↓	...
10. TGAGTCAB			3.1e-141	4.4e-165	↓	...

FIGURE 6.19 The motifs found by DREME.

DREME is deprecated by the developer. So, it is recommended to use MEME or STREME instead. Since we wish to find motifs in more than 50 FASTA sequences, we will use STREME program. STREME will attempt to find motifs by iterating five steps until the user-specified stopping criterion is met. The stopping criterion can be either the *p*-value of the motif or the total number of motifs found. The five steps include building of a suffix tree for both the ChIP-Seq FASTA sequences and control sequences, evaluation of the seed words, motif refinement, motif significance computation, and motif erasing (converting the site of the best motif to N in the primary sequences and control sequences).

The following STREME command searches for motifs in the FASTA sequences of the three ChIP-Seq samples:

```
streme --p chip1_peaks.fasta \
    --oc streme_motifs_chip1 \
    --dna \
    --thresh 0.05
streme --p chip2_peaks.fasta \
    --oc streme_motifs_chip2 \
    --dna \
    --thresh 0.05
```

TABLE 6.2 IUPAC Codes for DNA Alphabet

Nucleobase Name	IUPAC Symbol	Complement
Adenine	A	T
Cytosine	C	G
Guanine	G	C
Thymine	T U	A
Name	Symbol(s)	Matches
Any base	N . X	A C G T
Not T	V	A C G
Not G	H	A C T
Not C	D	A G T
Not A	B	C G T
Amino	M	A C
Purine	R	A G
Weak	W	A T
Strong	S	C G
Pyrimidine	Y	C T
Keto	K	G T



FIGURE 6.20 The motifs found by STREME.

```
streme --p chip3_peaks.fasta \
--oc streme_motifs_chip3 \
--dna \
--thresh 0.05
```

Each of the above commands will create a directory, “streme_motifs_*”. Four files will be produced inside each directory: “sequences.tsv”, “streme.html”, “streme.txt”, and “streme.xml”. Figure 6.20 shows how “streme.html” is displayed in the Internet browser.

HOMER is another motif discovery program that has an algorithm which performs de novo motif discovery in the regulatory regions of genes using a primary dataset and a secondary dataset. HOMER has two programs, `findMotifs.pl` and `findMotifsGenome.pl`, that perform the motif discovery in promoter and genomic regions, respectively. However, for ChIP-Seq enrichment sequences, we will use `findMotifs.pl` or `homer2`. Like MEME Suite programs, HOMER can also create a secondary random dataset if you do not have one. For the instructions of HOMER installation, visit "<http://homer.ucsd.edu/homer/introduction/install.html>". Homer includes a collection of perl and c++ programs designed to run in a UNIX/Linux environment. Refer to the website for the program requirements. On Ubuntu, after making sure that all required programs have been installed, you can create a directory "homer", download the installation Perl script, and install the HOMER as follows:

```
mkdir homer; cd homer
wget http://homer.ucsd.edu/homer/configureHomer.pl
perl configureHomer.pl -install
```

After the installation is complete, you will get a message as follows:

Add this line to your `.bash_profile` or `.bashrc` file (or other depending on your shell):

```
PATH=$PATH:/home/username/downloads/homer/./bin/
```

The path will depend on where you have installed HOMER. Copy the path and add it to the end of `.bashrc` file as follows:

```
export PATH=$PATH:/home/username/downloads/homer/./bin/:$PATH
```

You may need to restart the terminal or use "source `~/.bashrc`" for the change to take effect. To test HOMER installation, run "`findMotifs.pl`". That will display HOMER usage and option. The "`findMotifs.pl`" command requires your target sequences' file and a background sequence file in FASTA format specified by "`-fasta`" option. However, if you do not provide a background sequences' file, the command will use your target sequences to create background sequences. The general syntax for the use of "`findMotifs.pl`" command is as follows:

```
findMotifs.pl <targetSequences.fa> fasta <output directory> -fasta
<background.fa> [options]
```

We can use this command without providing background sequences' file as follows:

```
findMotifs.pl chip1_peaks.fasta fasta homer_motifs_chip1/
findMotifs.pl chip2_peaks.fasta fasta homer_motifs_chip2/
findMotifs.pl chip3_peaks.fasta fasta homer_motifs_chip3/
```

Three output directories will be created and several files are produced in each of the output directories:

The files “homerMotifs.motifs*” are the files that contain information of the motifs found by the de novo motif discovery method, separated by motif length (“*” simply represents an integer for the length of the motif sequence). The motif information includes the sequence, statistics, and PWM.

The file “homerMotifs.all.motifs” contains the information of all motifs found by the de novo method. It is the concatenated file made up of all the “homerMotifs.motifs*” files.

The file “motifFindingParameters.txt” contains the command line used to execute the program for the motif discovery.

The file “knownResults.txt” contains the statistics about known motif enrichment, including motif name, consensus, *p*-value, Log *p*-value, *q*-value (Benjamini), number of target sequences with motif, percentage of target sequences with motif, number of background sequences with motif, and the percentage of background sequences with motif.

The file “seq.autonorm.tsv” contains autonormalization statistics for the oligo.

The files “homerResults.html” and “knownResults.html” contain the output of de novo motif finding and known motif finding, respectively. These HTML files can be displayed on the Internet browser or using Firefox.

```
firefox homerResults.html knownResults.html
```

Figure 6.21 shows the motifs (logos and statistics) found by the de novo method using “findMotifs.pl”. The motifs are ordered by the *p*-value. The significant motifs must have very small *p*-value. Each motif has a link to the motif file.

The PWM motif discovery methods use prior information about the binding or interaction sites, obtained via laboratory means. Motif information are stored in a PWM file format that describes the probability of finding the respective nucleotides A, C, G, and T on each position of a motif. The word or PWM search method is used only to detect known binding sites of transcription factors. PWM files of known motifs are used to scan windows

Homer Known Motif Enrichment Results (homer/)												
Homer de novo Motif Results												
Gene Ontology Enrichment Results												
Known Motif Enrichment Results (txt file)												
Total Target Sequences = 39274, Total Background Sequences = 151140												
Rank	Motif	Name	P-value	log P-value	q-value	q-value (Benjamini)	# Target Sequences with Motif	% of Targets Sequences with Motif	# Background Sequences with Motif	% of Background Sequences with Motif	Motif File	SVG
1		ERG(ETS)/VCaP-ERG-ChIP-Seq (GSE14097) /Homer	1e-1550	-3.571e+03	0.0000		12257.0	33.79%	20766.6	13.74%	motif file (matrix)	svg
2		GAGA-repeat/SacCer-Promoters/Homer	1e-1293	-2.982e+03	0.0000		19842.0	54.70%	49248.5	32.59%	motif file (matrix)	svg
3		TF3A/C2H2/col-TF3A-DAP-Seq (GSE60143) /Homer	1e-1226	-2.825e+03	0.0000		9872.0	27.21%	16407.9	10.86%	motif file (matrix)	svg
4		Etv2(ETS)/ES-ER71-ChIP-Seq (GSE59402) /Homer	1e-1199	-2.762e+03	0.0000		8164.0	22.51%	11943.3	7.90%	motif file (matrix)	svg
5		PU.1-IRF(ETS)/IRF/Bcell-PU.1-ChIP-Seq (GSE21512) /Homer	1e-1096	-2.525e+03	0.0000		8612.0	23.74%	13894.9	9.20%	motif file (matrix)	svg
6		EHF(ETS)/LoVo-EHF-ChIP-Seq (GSE49402) /Homer	1e-1091	-2.513e+03	0.0000		9819.0	27.07%	17368.9	11.50%	motif file (matrix)	svg
7		Bcl6(Zf)/Liver-Bcl6-ChIP-Seq (GSE31578) /Homer	1e-998	-2.299e+03	0.0000		7806.0	21.52%	12440.0	8.23%	motif file (matrix)	svg
8		ETS1(ETS)/Jurkat-ETS1-ChIP-Seq (GSE17954) /Homer	1e-985	-2.268e+03	0.0000		9323.0	25.70%	16847.7	11.15%	motif file (matrix)	svg
9		SeqBias: GA-repeat	1e-982	-2.263e+03	0.0000		28361.0	78.18%	90409.3	59.84%	motif file (matrix)	svg
10		SCL(bHLH)/HPC7-Scl-ChIP-Seq (GSE13511) /Homer	1e-961	-2.215e+03	0.0000		21204.0	58.45%	59118.2	39.13%	motif file (matrix)	svg

FIGURE 6.21 The motifs found by HOMER.

of a sequence for detecting the presence of the known binding sites of interest. PWM files of known motifs can be download from motifs' database such as JASPAR at “<https://jaspar.genereg.net/>”. We will use MAST, which is one of the MEME Suite programs, to search for known motifs in our sequences. Assume that we wish to search for TATA binding site in our example sequences. First, we need to download the motif file from a database and then run the program as follows:

```
wget https://jaspar.genereg.net/api/v1/matrix/MA0108.1.meme
mast -mt 5e-02 \
    -oc mast_chip1 \
    MA0108.1.meme \
    chip1_peaks.fasta
```

Three output files (mast.html, mast.xml, and mast.txt) will be saved in the “mast_chip1” directory. You can use “firefox mast.html” to view the results.

6.4 SUMMARY

Identification of binding sites of proteins on the genomic DNA is critical for understanding gene regulation, pathways, and role of specific proteins in gene regulation and their implications of some diseases. Therefore, ChIP-Seq is used to study epigenetic change that affects gene expression and the impact of such changes on diseases. The ChIP-Seq is the most effective way to identify protein-binding sites on the genomic DNA. The binding sites of transcription factors and RNA polymerase II are found in the promoter regions of genes. In a ChIP experiment, the genomic DNA is cut into fragments. The DNA regions, where the protein of interest binds, are precipitated using a specific antibody. The protein molecules are then removed from the DNA fragments. The isolated DNA fragments are then sequenced using one of the sequencing techniques. The DNA library preparation and sequencing are similar to that of other sequencing applications. The sequence reads (in FASTQ files) produced by the sequencer are for the ChIP-Seq DNA reads that are likely to contain the binding sites for the protein of interest. The quality control step is carried out to reduce the error and to trim and remove adaptors and other technical sequences that may affect the analysis results. The cleaned reads are then aligned to a reference genome to produce BAM files that contain the alignment information of the ChIP reads. The unaligned, random, and mitochondrial reads are usually removed from the BAM files to reduce the computational burden. The peak enrichment regions, where the binding sites are most likely to be found, are called using one of the peak-calling programs. The peak information for each sample is saved in a BED file. We have used R Bioconductor package to visualize the distribution of the peaks and to perform annotation and functional analysis including GO and KEGG pathways. GO and KEGG enrichment analyses provide knowledge-based biological information. Finally, we used motif discovery programs to identify the motifs on the promoter regions.

REFERENCES

1. Bernstein BE, Meissner A, Lander ES: The mammalian epigenome. *Cell* 2007, 128(4):669–681.
2. Zhang Y, Liu T, Meyer CA, Eeckhoute J, Johnson DS, Bernstein BE, Nusbaum C, Myers RM, Brown M, Li W et al: Model-based Analysis of ChIP-Seq (MACS). *Genome Biol* 2008, 9(9):R137.
3. Narlikar L, Jothi R: ChIP-Seq data analysis: identification of protein-DNA binding sites with SISSRs peak-finder. *Methods Mol Biol* 2012, 802:305–322.
4. Heinz S, Benner C, Spann N, Bertolino E, Lin YC, Laslo P, Cheng JX, Murre C, Singh H, Glass CK: Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities. *Mol Cell* 2010, 38(4):576–589.
5. Ibrahim MM, Lacadie SA, Ohler U: JAMM: a peak finder for joint analysis of NGS replicates. *Bioinformatics* 2015, 31(1):48–55.
6. Feng J, Liu T, Qin B, Zhang Y, Liu XS: Identifying ChIP-seq enrichment using MACS. *Nat Protoc* 2012, 7(9):1728–1740.
7. Feng J, Liu T, Zhang Y: Using MACS to identify peaks from ChIP-Seq data. *Curr Prot Bioinfor* 2011, Chapter 2:Unit2 14–12.14.
8. Freese NH, Norris DC, Loraine AE: Integrated genome browser: visual analytics platform for genomics. *Bioinformatics* 2016, 32(14):2089–2095.
9. Yu G, Wang LG, He QY: ChIPseeker: an R/Bioconductor package for ChIP peak annotation, comparison and visualization. *Bioinformatics* 2015, 31(14):2382–2383.
10. Yu G, Wang LG, Han Y, He QY: clusterProfiler: an R package for comparing biological themes among gene clusters. *Omics* 2012, 16(5):284–287.
11. Bailey TL, Gribskov M: Combining evidence using p-values: application to sequence homology searches. *Bioinformatics* 1998, 14(1):48–54.



Taylor & Francis
Taylor & Francis Group
<http://taylorandfrancis.com>

Targeted Gene Metagenomic Data Analysis

7.1. INTRODUCTION TO METAGENOMICS

We can use any of the sequencing applications discussed in the previous chapter to study an individual bacterium. Rather than a single species of bacteria, metagenomics involves studying the genomes of a community of bacteria recovered from environmental or clinical samples to obtain a variety of knowledge of the microbial species present in the sample and their impacts on other living organisms. The most common samples targeted by metagenomics include human skin and cavities, digestive system, water, plants, soil, waste (liquid and solid, feces), and food products. Metagenomics has a variety of uses including identification of an unknown pathogen in outbreaks of a disease, clinical diagnosis, monitoring human and animal health, identification of bioactive compounds (terragines, violacein, and indirubin) [1], drugs from marine microorganisms such as cytarabine (anti-cancer) [2], cephalosporins (anti-microbial) [3], and vidarabine (anti-virus) [2], discovery of novel antibiotics, production of some enzyme (lipases, proteases, lyases, amylases, etc.), exploring new industrial and healthy products (indigo, probiotics) [4], and investigation and monitoring of wildlife health.

In a typical metagenomics study, genomic DNA of the whole bacterial community in a sample is extracted and then sequenced, and a pipeline of analyses is conducted on the acquired sequence data. There are two sequencing approaches to characterize microbial taxonomic groups in environmental samples. The first one targets a specific marker gene or a region of a marker gene after being amplified with polymerase chain reaction (PCR). The 16S rRNA gene is often used for this purpose. The second approach targets all bacterial genomes in the samples and uses the shotgun whole genome sequencing to achieve that. We will discuss the shotgun sequencing in the next chapter. In this chapter, we will focus on the amplicon-based sequencing to identify bacteria in environmental or clinical samples.

The analysis of the metagenomics data involves classification of an individual sequence to a bacterial taxon or taxa. Bacteria are classified into hierarchical taxonomic groups in descending order (kingdom, phylum, class, order, family, genus, species, and subspecies). The analysis also includes construction of phylogenetic tree for the bacterial community, quantification of bacterial presence in the sample (abundance), and the microbial diversity in an individual sample and across samples. A number of bioinformatics tools are available for analyzing metagenomic data.

In the amplicon-based metagenomics, the targeted region is a marker gene or any part of a gene or any genomic region appropriate for microbial identification. The ideal target region is the one that includes a highly conserved sequence surrounded by less conserved and it must be present ubiquitously in all target species and with available reference sequences in the sequence databases. The 16S rRNA gene is one of the best candidate marker genes. The 16S rRNA gene codes for a component of the 30S small subunit of the bacterial ribosome. It is around 1500 bp consisting of nine conserved regions surrounding hyper-variable regions. The sequences of the conserved regions are labeled as C1, C2, ..., C9 and the variable regions are labeled as V1, V2, ..., V9. PCR primers are designed from the conserved region close to the variable region so that the species-specific targeted region is amplified and enriched by the PCR amplification. Then, the amplicons are sequenced and analyzed.

Several samples are usually sequenced in a single run using multiplexing approach in which unique barcode sequences are ligated to the DNA of each sample in the library preparation step. After sequencing, the reads are demultiplexed by separating the reads of the individual samples into separate FASTQ files before analysis. Since the amplicon-based metagenomics depends on a targeted region of the genomes, it has less resolution than the shotgun whole genome sequencing but is less expensive.

7.2 ANALYSIS WORKFLOW

In the following, we will discuss the steps of the workflow of the amplicon-based metagenomics data analysis, which include raw data preprocessing, read clustering, denoising (error removal), taxonomic group assignment, construction of phylogenetic tree, and diversity analysis.

7.2.1 Raw Data Preprocessing

After sequencing the targeted marker, raw sequence data is obtained in FASTA or FASTQ format. In the case of FASTA format file produced by Sanger sequencing method, the per base quality score may be provided in a separate file. The format of FASTQ files allows base quality scores to be in the same file. The base quality scores reveal base call quality of each base and enable us to assess the sequence reads and to determine if the reads require preprocessing before the analysis. The quality control step should not be taken lightly. Errors in base calls may occur due to the library preparation or sequencing. The reads of low-quality scores are usually filtered or truncated so that the errors do not affect the final results. The preprocessing of the raw sequence data may also include demultiplexing if multiple samples are sequenced in a single run. The demultiplexing step depends on

how the reads and barcodes are stored. The reads also might be sequenced as single end or paired end. The paired-end reads are usually stored in two FASTQ files.

7.2.2 Metagenomic Features

After the preprocessing, the raw reads of the targeted gene (16S rRNA) are dereplicated by reducing the abundance of the similar reads based on a Hamming distance threshold. The dereplicated reads are then clustered by collapsing the complete sets of reads into a grouping units called operational taxonomic units (OTUs) [5]. Once the OTUs have been formed, representative sequences (features) can either be selected from those OTUs or the sequences of each OTUs are denoised to remove potential errors before selecting representative sequences. In the following, we will discuss both clustering and denoising.

7.2.2.1 *Clustering*

There are three approaches for clustering the preprocessed amplicon reads. The first is the de novo clustering [6] in which the reads are clustered into OTUs according to their similarity at a specified threshold. It compares each read against each other and then it groups sequences into OTUs. The second is called open-reference clustering in which reads are clustered around previously annotated reference sequences by sequence classification or searching in a database (Greengenes database) and the reads that do not cluster around reference sequences are clustered with de novo method. The third is the closed-reference clustering in which reads are clustered around reference sequences and the reads that do not find reference sequences are removed [7]. The closed-reference clustering may introduce reference bias to the clustering process since the reads without references will be removed, while they may represent novel species or taxa. The de novo clustering is preferred, although it may be computationally expensive other than the other two clustering methods. The algorithms for de novo clustering include hierarchical clustering and heuristic clustering. The hierarchical clustering method requires a matrix for the pairwise distances between all unique sequences. A hierarchical tree is constructed from the distance matrix, and then the OTUs are constructed from the tree based on a distance threshold.

The heuristic clustering uses pairwise sequence alignment to construct a distance matrix one by one instead of computing all distances in a single step. The heuristic clustering method generates OTUs in a greedy incremental strategy that utilizes one sequence as a seed to represent a cluster. Then, each one of the unique reads is compared with the seeds of existing clusters. A read is assigned to a cluster if the pairwise distance between a unique read and a seed meets a predefined threshold. If a read does not join any cluster, that read will become a seed for a new cluster. The final clusters represent the OTUs. The heuristic clustering method is more computationally efficient than the hierarchical clustering. There are a variety of heuristic clustering methods that vary in seed selection and distance calculation. Examples of the tools that use heuristic clustering for constructing OTUs include USEARCH and CD-HIT.

In general, an OTU constructed by any of the clustering methods is considered as a taxonomic group. From this point, the analysis can move on to the taxonomic group assignment, construction of the phylogenetic tree, and diversity analysis but the latest analytic

tools proposed an extra step before taxonomic assignment. This step is meant to reduce the potential errors that may produce noises; therefore, the step is called denoising.

7.2.2.2 Denoising

There are two possible types of errors that may occur on deciding whether the variation within an OTU represents errors or real diversity. The first type is the base calling error which may arise from the sequencing. This type of errors may occur due to the incorrect base pairing during the PCR amplification, polymerase slippage, or PCR chimeras that are formed when the DNA strand extension is aborted during the PCR process and the aborted products act as primers in the next PCR cycle producing artifacts. The second type of errors is the misclassification of a read to an incorrect taxonomic group. This error can be corrected by constructing OTUs at a particular similarity threshold such as 97%. However, that may come at the cost of taxonomic sensitivity. Denoising is attempting to handle these errors by using the reads to infer the correct biological sequences. This way the misclassification can be avoided.

Several computational approaches have been proposed for sequence denoising. The most commonly used approaches are DADA2, Deblur, and UNOISE3 which are able to infer error-free biological sequences at a single-nucleotide resolution. Those inferred sequences that will be used for taxonomic assignment are called features, zero-radius OTUs (ZOTUs), exact sequence variants (ESVs), or amplicon sequence variants (ASVs). In the following, we will discuss those three popular denoising methods.

7.2.2.2.1 DADA2 Denoising

DADA2 (Divisive Amplicon Denoising Algorithm 2) [8] was adapted to use with Illumina sequencing and available as an open-source R package and as plugin in QIIME2, which is an open-source command-line Linux program. DADA2 implements a new model of Illumina-sequenced amplicon errors that incorporates quality information of the within-sequence errors and between-sequence errors. The model quantifies the error rate (λ) at which an amplicon read is produced from a sample sequence as a function of a sequence composition and quantity. The number of amplicons or abundance follows the Poisson distribution with the parameter λ_{ji} , which is the error rate at which an amplicon read with sequence i is produced from sample sequence j . The abundance of the sequence i has an expected value equal to an error rate λ_{ji} multiplied by the expected reads of sample sequence j .

The DADA2 model assumes that errors occur independently with a read and independently between reads. The model then estimates the error rate as the product over the transition probabilities between the L aligned nucleotides and associated quality score of the original nucleotide as follows:

$$\lambda_{ji} = \prod_{l=0}^L p(j(l) \rightarrow i(l), q_i(l)) \quad (7.1)$$

where $p(j(l) \rightarrow i(l), q_i(l))$ is the transition probability between aligned nucleotides $j(l)$ and $i(l)$ and the associated quality score $q_i(l)$, e.g., $p(T \rightarrow G, 40)$.

The divisive partitioning algorithm begins with all amplicon reads in a single partition. The error rate is then used to model the number of observed reads of each unique sequence to compute the p-value of the hypothesis that the number of amplicons of each unique sequence is consistent with the error model. According to the DADA2 model, for the unique sequence i with abundance a_i be in partition j containing n_j reads, the abundance p-value is given as

$$p_A(j \rightarrow i) = \frac{1}{1 - p_{\text{poisson}}(n_j \lambda_{ji}, 0)} \sum_{a=a_i}^{\infty} p_{\text{poisson}}(n_j \lambda_{ji}, a) \quad (7.2)$$

These p-values of the unique sequences are used as the division criteria for an iterative partitioning. A threshold is specified for partition; if the smallest abundance p-value falls below the threshold, a new partition is formed with that unique sequence allowing other similar unique sequences to join it. The division continues iteratively until all unique sequences falling within a OTUs are consistent with abundance p-values greater than the specified threshold.

The output of the divisive amplicon denoising algorithm is a collection of ASVs, which are exact sequences with defined statistical confidence. Because ASVs are exact sequences, generated without clustering or reference databases, they can be readily compared between studies using the same target region. DADA2 pipeline generates an ASV table that can be used for downstream analysis.

7.2.2.2.2 Deblur Denoising

Deblur [9] is a denoising method that uses error profiles of amplicons sequenced by Illumina MiSeq and HiSeq sequencing platforms to infer error-free sequences. Unlike DADA2, Deblur operates on each sample independently. The Deblur algorithm begins by comparing the pairwise Hamming distances of all sequences within a sample to an upper-bound error profile. The unique sequences are sorted by abundance by ascending from the most to the least. Neighboring reads are formed for each read based on a Hamming distance threshold. The number of incorrect reads is then subtracted from the abundance of the neighboring reads using an upper bound on the error probability. After subtraction, the sequences with zero abundance are considered as a noise and dropped from the list of the valid sequences. Deblur can infer the correct sequences. However, it may decline to remove PCR chimeras that are produced from the aborted PCR cycle.

7.2.2.2.3 UNOISE2 Denoising

Unlike DADA2, UNOISE [10] denoising does not use quality score and it utilizes one-pass clustering strategy with only two parameters (α and β) with pre-set values. A unique read sequence (M) in a cluster is evaluated based on its Levenshtein distance (d) from

the centroid sequence (C) and skew of its abundance (a_M) with respect to the centroid sequence abundance (a_C), which is given as

$$\text{skew}(M, C) = \frac{a_M}{a_C} \quad (7.3)$$

When a member unique sequence has both an enough small distance and an enough small skew with respect to the centroid sequence, then it is likely that sequence is incorrect read of the centroid sequence with d points errors. The maximum skew (β) allowed for a cluster member with d differences from the centroid sequence is given by

$$\beta(d) = \frac{1}{2^{\alpha d + 1}} \quad (7.4)$$

where α is set to 2 by default.

We can notice that as the distance d between the member sequence and centroid increases, the maximum skew β decreases exponentially.

The unique sequences with low abundance are removed by the UNOISE2 algorithm.

The final products of any of the clustering and denoising methods are feature table and the list of representative sequences. The feature table provides the feature abundance or the number of a times a feature has been observed in a sample. A feature is a unit of observation that can be an OTU or an ASV. The feature table is needed for the downstream analysis such as taxonomy assignment, construction of phylogenetic tree, and diversity analysis.

7.2.3 Taxonomy Assignment

Given a set of representative sequences generated by the above-discussed clustering or denoising methods, the taxonomy assignment step will attempt to assign taxa for each sequence. There are several methods for assigning taxonomy but, in general, they can be categorized into (i) alignment-based methods such as BLAST and VSEARCH and (ii) machine learning methods such as Ribosomal Database Project (RDP) Classifier. The output of the taxonomy assignment methods is mapping a representative sequence to taxa and providing an assignment quality score.

7.2.3.1 Basic Local Alignment Search Tool

The Basic Local Alignment Search Tool or BLAST [11] is a widely used seed-based heuristic sequence search tool whose algorithm is adopted from the Smith-Waterman algorithm for local sequence alignment. Providing a representative sequence (generated from clustering or denoising) as a search query to BLAST, the search is conducted against a database of sequences with known taxonomy. Rather than aligning to a single sequence, the taxonomy assignment is based on the consensus of hits in the reference database that exceed the predetermined percent identity. If the blast hits agree on the same taxonomy, then the representative sequence will be given that taxonomy level with consensus greater than a threshold.

7.2.3.2 VSEARCH

VSEARCH [12] is another alignment-based and open-source tool like BLAST. However, unlike BLAST, which uses seed-based heuristic search algorithm, it uses a fast heuristic search by identifying a small set of database sequences that have many k-mer words in common with the query sequence (a representative sequence). VSEARCH algorithm counts the number of shared words between a representative sequence and each database sequence (a word will count once if it appears multiple times). Thus, the similarity between a representative sequence and a target sequence is based on the statistics of shared words rather than alignment. Then, the algorithm performs pairwise global alignment (Needleman-Wunsch) for the query sequence with each sequence of the database beginning with the sequences with the largest number of words in common with the representative sequence. The optimal global alignment score is computed for each alignment. Unlike BLAST, exhaustive pairwise alignment is computationally expensive. However, VSEARCH employs parallel strategies by using vectorization and multiple threading that reduce the computational cost.

7.2.3.3 Ribosomal Database Project

Ribosomal Database Project (RDP) [13] Classifier is a machine learning taxonomy classification method. It is a model trained by known reference sequences with known taxonomic groups. After training, the classifier is used to predict the taxonomy of the representative sequences. RDP uses the naïve Bayesian algorithm to accurately predict the taxonomy of a sequence. The model is trained by N sequences with known taxonomy. First, the algorithm forms a dictionary from all possible 8-base subsequences or words from all sequences. The words of all N sequences are called corpus. Let $W = \{w_1, w_2, \dots, w_d\}$ be the set of all possible eight-character words in the corpus and $n(w_i)$ be the number of sequences containing word w_i , where $i=1, 2, \dots, d$; thus, the expected-likelihood estimate of observing word w_i in a sequence is calculated as

$$P_i = \frac{[n(w_i) + 0.5]}{(N+1)} \quad (7.5)$$

The numbers 0.5 and 1 are used to keep the value of P_i to be between 0 and 1 (e.g., 0 $P_i < 1$).

Assume that $m(w_i)$ is the number of the word w_i contained by M training sequences whose genus is G . Thus, the conditional probability that a sequence of genus G contains the word w_i is estimated as:

$$P(w_i | G) = \frac{[m(w_i) + P_i]}{M+1} \quad (7.6)$$

Now assume that a sequence S of genus G has the observed set of words $V = \{v_1, v_2, \dots, v_f\} \subseteq W$; thus, the joint probability or likelihood of observing the sequence S is the product of the probability of each word given genus G as

$$P(S|G) = \prod P(v_i|G) \quad (7.7)$$

Finally, the probability that an unknown query sequence S is a member of genus G is given as

$$P(G|S) = P(S|G) \times \frac{P(G)}{P(S)} \quad (7.8)$$

where $P(G|S)$ is the posterior probability, $P(G)$ is the class prior probability of a sequence being a member of G , and $P(S)$ is the predictor prior probability.

Since both class prior probability and predictor prior probability are constant for all sequences, we can drop both $P(G)$ and $P(S)$. So, the above formula can be rewritten as

$$P(G|S) = P(S|G) = \prod P(v_i|G) \quad (7.9)$$

Based on the naïve Bayesian, each representative sequence obtained from the denoising step is assigned to the genus G or a taxonomic group that is giving the highest probability score after calculating the score for each taxonomic group.

7.2.4 Construction of Phylogenetic Trees

A phylogenetic tree is a diagram that represents evolutionary relationships among organisms. The pattern of branching in a phylogenetic tree describes how species or taxonomic groups evolved from a series of common ancestors and how they are related. Two species are more related if they have a more recent common ancestor and less related if they have a less recent common ancestor. In amplicon-based metagenomic studies, phylogenetic trees based on targeted genes (e.g., 16S rRNA gene) are commonly used to describe and compare the taxonomic groups of the bacterial community in the sample. There are several tree-building algorithms implemented in software. The construction of the phylogenetic tree begins with computing genetic distances between the unique representative sequences identified in the above steps. The genetic distance between two representative sequences is the number of mutations or evolutionary events between two sequences since their divergence. The traditional Hamming method calculates genetic distance between any two sequences by counting the number of differences between them. However, this method does not capture the evolutionary history of the sequences since not all mutations are shown in the current sequences. The Hamming distance is corrected with Jukes–Cantor logarithmic correction. Let p be the distance between two sequences, the distance (d) after Jukes–Cantor correction is given as

$$d = \frac{3}{4} \log \left(1 - \frac{4}{3} p \right) \quad (7.10)$$

The tree is then built from the pairwise distances of the representative sequences. Several tree-building methods are available. The most popular distance-based methods are the

unweighted pair group method with arithmetic mean (UPGMA), weighted pair group method with arithmetic mean (WPGMA), and neighbor joining (NJ) [14].

Both UPGMA and WPGMA assume a randomized molecular clock that measures the evolutionary divergence of sequences. The molecular clock is defined as the average rate at which a sequence accumulates mutations. Both UPGMA and WPGMA also have a similar algorithm. They use a cluster procedure that assumes each representative sequence as a cluster on its own and then they join the closest clusters and recalculate the distance of the joint pair by the average. These steps are repeated until all sequences are connected in a single cluster. However, the difference between the two methods is that in UPGMA, equal weight is assigned on the distances, while in the WPGMA different weights are assigned on the distances.

The algorithm of the NJ method does not make an assumption of the molecular clock and it adjusts for the rate variation among branches. The algorithm begins with an initial unsolved star-like tree made up of the representative sequences. The distance between each pair is evaluated. The first joint is created by joining the closest two neighboring sequences and a branch is inserted between them and the rest of the star-like tree. The value of the branch is recalculated on the basis of their average distance. This process is repeated until only one terminal is present from the initial tree.

The above briefly described tree construction methods are distance-based and less computationally expensive. However, there are other methods including maximum parsimony (MP) and maximum likelihood (ML) which make use of all known evolutionary information (individual substitutions) to determine the most likely ancestral relationships. Refer to a book for phylogenetic tree for more details about the various tree construction methods.

A phylogenetic tree is either rooted (with a common ancestor for all sequences) or unrooted (without common ancestor). The unrooted trees are constructed when we do not make the assumption that the molecular clock evolution is valid and they only reflect the relationship among representative sequences but not the evolutionary path. However, if we can make the assumption that sequences evolve at rates that remain constant through time for different lineages, then the root of a tree is estimated as the midpoint of the longest span across the tree.

7.2.5 Microbial Diversity Analysis

The microbial diversity or richness is calculated from the feature table, obtained in the denoising step above, to describe the number of different species of microbes present within individual samples and between samples. The diversity of the microbial community within a sample is called alpha diversity, while the measure of similarity or dissimilarity of microbial communities in two samples is called beta diversity. For alpha diversity, there are several diversity metrics including Shannon's diversity index, observed features, Faith's phylogenetic diversity, and evenness. The beta diversity metrics include Jaccard distance, Bray–Curtis distance, and unweighted UniFrac distance.

7.2.5.1 Alpha Diversity Indices

7.2.5.1.1 Shannon's Diversity Index

The Shannon's Index also called Shannon–Wiener Index is the most commonly used measure for alpha diversity of an individual sample. It is defined as:

$$H = \sum_{i=1}^S p_i \times \ln(p_i) \quad (7.11)$$

where p_i is the probability of finding species/taxon i in the sample and S is the number of taxa in the sample or richness.

The values of Shannon diversity usually fall between 1.5 and 3.5. The higher the value of H , the higher the diversity of taxa in a sample. The lower the value of H , the lower the diversity. A sample with a single taxon will have $H=0$ (not diverse sample).

7.2.5.1.2 Pielou's Evenness

The evenness of taxa in a sample refers to how similar the abundances of the different taxa in the sample are and it is obtained by dividing Shannon–Wiener Index by the natural logarithm of the number of unique taxa in the sample as follows:

$$\text{Evenness} = \frac{H}{\ln(S)} \quad (7.12)$$

The evenness value ranges from 0 to 1, where 1 indicates complete evenness or the taxonomic groups in the sample have similar abundances.

7.2.5.1.3 Faith's Phylogenetic Diversity Index

Faith's phylogenetic diversity (PD) index is a qualitative measure of community richness. It is based on phylogeny and is defined as the phylogenetic diversity of a set of sequences being equal to the sum of the lengths of all the branches on the tree. A higher PD indicates more richness or diversity in the sample.

7.2.5.2 Beta Diversity

Beta diversity quantifies the difference among biological communities in different samples. It is defined as the ratio between regional (gamma) and local (alpha) diversities. Most of the beta diversity metrics are based on dissimilarity measures such as Jaccard distance, Bray–Curtis distance, and weighted and unweighted UniFrac distance. Let a be the number of species/taxa found in both samples, b is the number of species/taxa in the first sample but not in the second sample, and c is the number of species/taxa in the second sample but not in the first sample. The beta diversity metrics are computed as follows.

7.2.5.2.1 Jaccard Diversity Distance

$$\beta_{\text{jac}} = \left(\frac{b+c}{a+b+c} \right) \times 100 \quad (7.13)$$

If the samples share all the taxonomic groups, the percent Jaccard index will be 100% similar ($\beta_{\text{Jac}} = 100\%$); the closer it is to 100%, the more similar the samples are. If the two samples share no species/taxa, they will be 0% similar ($\beta_{\text{Jac}} = 0$). If the percent Jaccard index is 50%, the two sample will share half of the taxonomic groups.

7.2.5.2.2 Bray–Curtis Dissimilarity Index

$$\beta_{\text{br}} = \left(1 - \frac{2c}{a+b} \right) \times 100 \quad (7.14)$$

The percent Bray–Curtis dissimilarity is always a number between 0 and 100. If it is 0, then the two samples share all the same species; if it is 100, that means the two samples do not share any species.

7.2.5.2.3 Unweighted and Weighted UniFrac Distance Index

The UniFrac is phylogenetic-based beta diversity index that takes into account the evolutionary relatedness of the communities in the two samples. The UniFrac distance index is defined as the fraction of the observed branch lengths of the phylogenetic tree that is unique in either sample. If the communities of the two samples are identical, UniFrac index will be zero. If the two communities are evolutionarily unrelated, the UniFrac index would be 1.0. The UniFrac index is closer to zero if the communities of the two samples are more evolutionarily related. The weighted UniFrac uses relative abundances of species in the samples as a weight on the branch lengths (thus, it emphasizes the dominant species). While unweighted UniFrac uses only presence or absence (thus, it emphasizes the rare species).

7.3 DATA ANALYSIS WITH QIIME2

Now it is time to get your hand dirty with some worked examples that cover raw data pre-processing, read clustering, denoising, taxonomic assignment, phylogenetic tree, and diversity analysis. For this purpose, we will use QIIME2 (Quantitative Insights Into Microbial Ecology 2) [15], which is the most commonly used free program for analysis of amplicon-based microbial sequencing data. QIIME2 can be used for any analysis of any targeted gene sequencing data but the program modules for the analysis of metagenomic data based on 16S rRNA gene are very well established. QIIME2 can be installed in different platforms. For the detailed installation instructions, visit "<https://docs.qiime2.org/2022.2/install/>". If you have Anaconda installed on your Linux computer, you can install it with "conda install -c qiime2 qiime2"; however, make sure that all requirements are met. We will use QIIME2 under the Anaconda environment. After installing QIIME2 under Anaconda, run "conda activate qiime" on the Linux terminal to activate QIIME2 environment. Once it has been activated, the terminal prompt will change into something like "(qiime2)\$". Then, you can run any QIIME command. To display the available QIIME2 commands, run the following:

```
(qiime2)$ qiime
```

This will display the program usage, options, and available commands. A command can be a QIIME2 utility tool (info, tools, dev) or QIIME2 plugins. We will use the “tools” command very often. To use a QIIME2 command or plugin, it must be in the “qiime <command>”. For instance, to use “tools” command, run the following:

```
(qiime2)$ qiime tools
```

The QIIME2 plugins are software packages developed by programmers to be used for analysis. Plugins work through two functional units: methods and visualizers. Before any data processing, QIIME2 converts input raw data into a special format called QIIME2 artifacts. An artifact is a compressed file with the “.qza” file extension. For the output, QIIME2

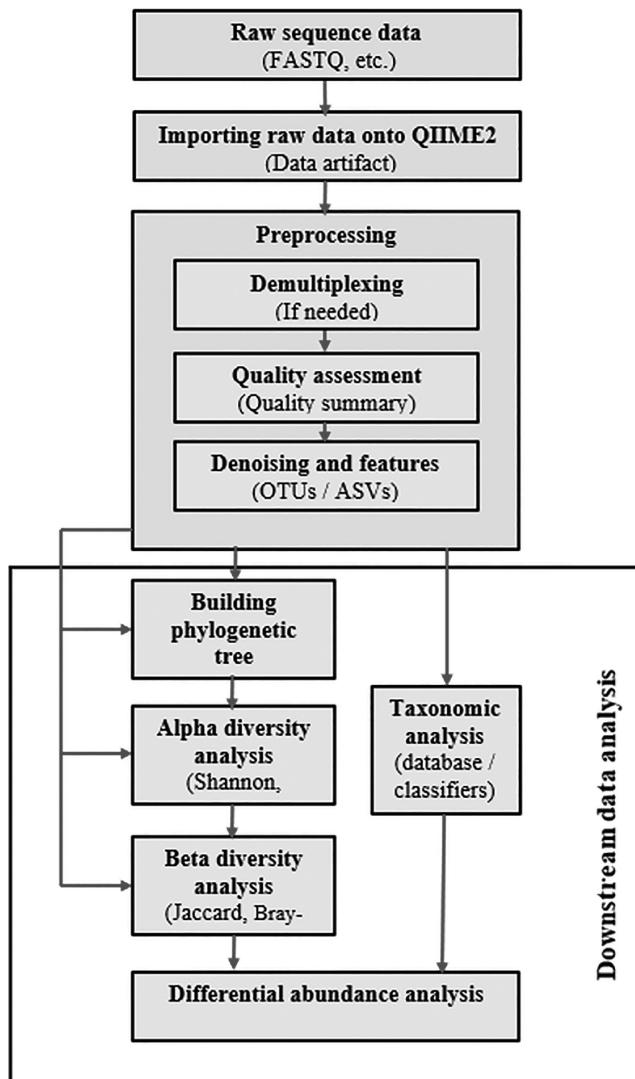


FIGURE 7.1 General QIIME2 workflow for amplicon-based sequence data analysis.

generates another file format, with the “.qzv” file extension, called visualization file. This visualization file is a standalone and sharable file that may contain any kinds of output such as images, tables, and interactive representations. Plugin methods take QIIME2 artifacts as input and produce an output. While a plugin visualizer produces a single visualization file for the purpose of visualizing or sharing. In the following, we will show you how to use QIIME2 to analyze targeted gene metagenomic data. The general workflow of the amplicon-based metagenomic data analysis is shown in Figure 7.1.

7.3.1 QIIME2 Input Files

Metagenomic amplicon-based raw data may be acquired from a study conducted at the laboratory or may be downloaded from a database. A study will have a design tailored to address research objectives. The study design usually dictates the workflow of the analysis. Whether raw data is from your own project or downloaded from a database, it usually includes (i) raw sequence data and (ii) metadata (information of the samples and study design). The analysis with QIIME2 requires importing these two inputs and converting them into artifacts. Then, only the artifacts are the files that are used for the analysis. Therefore, the first task is to import the raw sequence data and metadata into artifacts. The QIIME2 artifacts have their semantic type that enables QIIME2 to identify the suitable artifact for an analysis. In the following, we will discuss the raw sequence and metadata with more details.

7.3.1.1 Importing Sequence Data

QIIME2 accepts inputs in a variety of file formats including FASTA, FASTQ files, and feature files of OTUs or representative sequences. The FASTQ (single-end or paired-end) files are the most commonly used. The reads in FASTQ files may be multiplexed or demultiplexed. If they are multiplexed, they can either be multiplexed following the Earth Microbiome Project (EMP) protocol (the barcode sequences are in a separate file) or non-EMP (the reads are with in-sequence barcodes) [16]. On the other hand, the demultiplexed reads can either be in Casava 1.8 format or not.

Some laboratories may have their own sequencer and others may depend on genomic core facilities for sequencing. In either case, the raw data would be provided in one of the above formats. Raw sequence data can also be downloaded from metagenomics databases. Examples of metagenomic databases include NCBI SRA database available at “<https://www.ncbi.nlm.nih.gov/sra>” and the EMBL-EBI-hosted MGnify database available at “<https://www.ebi.ac.uk/metagenomics/>”. Both databases provide data generated from a variety of microbiome studies on specific environment such as human body sites, soil, seawater, and others. MGnify microbiome data can also be accessed from the NCBI SRA. Sequence data like FASTQ files generated from those studies are stored in the NCBI SRA as sequence read archives, which are compressed files and can be downloaded using SRA-toolkit.

Whether the input file for QIIME2 is FASTA file, FASTQ files, or feature file, it must be imported by QIIME and converted into QIIME2 artifact. Importing an input file into an artifact depends on the raw data file format; each file format is imported in a unique way. But, in general, to import any input data, you must use “qiime tools import”. We already

know that to execute any QIIME2 command, you must use “qiime”. We also mentioned above that “tools” is a command and command or plugin may have methods or visualizers or both. The “import” above is a tools command method whose function is to import any input data. However, for each input file format, different arguments and options will be used. In general, importing an input file into an artifact will follow this format:

```
qiime tools import \
  --type xxx                      #raw data type
  --input-path xxx                  #the directory
  --input-format xxx                #the raw data format
  --output-path xxx                #where artifact file is to be saved
```

The “import” options are self-explanatory, where “--type” specifies the type of the data to be imported, “--input-path” specifies the path to the raw data files, “--input-format” specifies the input file format, and “--output-path” specifies the path where the artifact file will be saved.

You must specify the right data type and format on “--type” and “--input-format” for your input files to be imported successfully.

In the following, we will show you the formats for importing different output files, but for practice and detailed information, visit “<https://docs.qiime2.org/>”.

We will discuss the import of the common types of raw data below.

7.3.1.1.1 Importing Raw Data in FASTA

If the raw data is in FASTA format, all sequences must be in a single FASTA file consisting of exactly two lines for each sequence record; a line for the definition line and a single line for sequence (not in multiple lines). The ID in each in the definition line must be as per the FASTA specification (no white space between the “>” and the ID). Moreover, the ID must follow the “SAMPLEID_SEQID” format, where SAMPLEID is a unique sample identifier and SEQID is a sequence identifier. Assuming that your sequences are in the “sequences.fasta” file, you can use the following “qiime tools import” to create an artifact for the input file:

```
qiime tools import \
  --type 'FeatureData[Sequence]' \
  --input-path sequences.fna \
  --output-path sequences.qza
```

The above command will import the FASTA sequences as a QIIME2 artifact “sequences.qza” that is ready for the next step of the analysis.

7.3.1.1.2 Importing EMP-Multiplexed FASTQ Reads

The EMP-multiplexed FASTQ files contain multiplexed reads, but the barcode sequences are in a separate file. Therefore, there will be two files (forward FASTQ file and barcode FASTQ file) for single-end reads and three files for paired-end reads (forward, reverse,

and barcode FASTQ files). We can import such multiplexed raw data to QIIME2 and later demultiplex them using a QIIME2 command. To import a single-end EMP-multiplexed reads, the forward FASTQ file name must be “sequences.fastq.gz” and the barcode file is to be “barcodes.fastq.gz”. These two files can be in a directory say “data” for example. Then, you can use “qiime tools import” to import the raw data using the following:

```
qiime tools import \
--type EMPSingleEndSequences \
--input-path data \
--output-path artifacts/multiplexed-emp-single-end.qza
```

Notice that we use “input-path” to specify the directory where the two files are found.

For the paired-end EMP-demultiplexed raw data, we can use the following:

```
qiime tools import \
--type EMPPairedEndSequences \
--input-path data \
--output-path artifacts/multiplexed-emp-paired-end.qza
```

Notice that both artifacts created by the above commands are for multiplexed reads. Those artifacts required demultiplexed as we will discuss later.

7.3.1.1.3 Importing non-EMP-Multiplexed FASTQ Files

In the non-EMP-multiplexed reads, the barcode sequences are attached to the reads. This read format will have a single FASTQ file (“sequences.fastq.gz”) for the single-end reads and two files (“forward.fastq.gz” and “reverse.fastq.gz”) for the paired-end reads. The following are the commands to import single-end and paired-end raw data, respectively:

```
qiime tools import \
--type MultiplexedSingleEndBarcodeInSequence \
--input-path data/sequences.fastq.gz \
--output-path artifacts/multiplexed-single-end.qza

qiime tools import \
--type MultiplexedPairedEndBarcodeInSequence \
--input-path data \
--output-path artifacts/multiplexed-paired-end.qza
```

Again, the artifacts created by the above commands contain multiplexed reads. So, they must be demultiplexed before proceeding. Demultiplexing will be discussed later.

7.3.1.1.4 Importing Casava Demultiplexed FASTQ Files

The Casava 1.8 demultiplexed formatted FASTQ file includes a barcode as the sample identifier in the FASTQ file name, which is made of four IDs separated by underscore: the sample identifier, barcode identifier, slide lane number, direction of the read, and the set

number. QIIME2 can use that information in the file name to automate importing the files and to create an artifact for that demultiplexed raw data. The following are the commands for importing Casava 1.8 formatted demultiplexed, single-end and paired-end FASTQ files, respectively:

```
qiime tools import \
--type 'SampleData[SequencesWithQuality]' \
--input-path data \
--input-format CasavaOneEightSingleLanePerSampleDirFmt \
--output-path artifacts/demultiplexed-single-end.qza

qiime tools import \
--type 'SampleData[PairedEndSequencesWithQuality]' \
--input-path data \
--input-format CasavaOneEightSingleLanePerSampleDirFmt \
--output-path artifacts/demultiplexed-paired-end.qza
```

These commands create artifacts for demultiplexed data that are ready for processing (the samples are already separated).

7.3.1.5 Non-Casava Demultiplexed FASTQ

When the reads of each sample are in a separate FASTQ file for single-end reads or in two FASTQ files for the paired-end reads, these demultiplexed files are non-Casava 1.8 demultiplexed FASTQ files. Those are the most commonly used files. In our worked example, we will practice in files with this format. Since the analysis of metagenomic data involves multiple samples, QIIME2 will need to know the sample identification when multiple FASTQ files for individual samples are used in the analysis. Thus, importing the non-Casava 1.8 demultiplexed FASTQ files requires an additional text file that maps sample identifiers to the FASTQ files. This file is called the manifest file and it will be created by the user. A manifest file is a comma-separated value (CSV) file with three columns with these column headers: “sample-id”, “absolute-filepath”, and “direction”. The headers are self-explanatory; the sample-id column includes the sample IDs, the absolute-filepath includes the absolute path for a FASTQ file, and the direction column shows the direction of the FASTQ file (forward or reverse). Figure 7.2 shows example manifest files for single-end FASTQ files (on the left) and paired-end FASTQ files (on the right).

When you have multiple FASTQ files, each file (or two files) for an individual sample, you will need to create a manifest file for importing those FASTQ files into QIIME2 artifact. The single-end FASTQ files and paired-end FASTQ files are imported, respectively, as follows:

```
qiime tools import \
--type 'SampleData[SequencesWithQuality]' \
--input-format SingleEndFastqManifestPhred33 \
--input-path data \
--output-path artifacts/xxxx.qza
```

<pre>sample-id,absolute-filepath,direction sampleID1,/home/path/sample1.fastq,forward sampleID2,/home/path/sample2.fastq,forward sampleID3,/home/path/sample3.fastq,forward sampleID4,/home/path/sample4.fastq,forward</pre>	<pre>sample-id,absolute-filepath,direction sampleID1,/home/path/sample1_1.fastq,forward sampleID1,/home/path/sample1_2.fastq,reverse sampleID2,/home/path/sample2_1.fastq,forward sampleID2,/home/path/sample2_2.fastq,reverse</pre>
Ln 6, Col 1 100% Unix (LF) UTF-8	Ln 1, Col 1 100% Unix (LF) UTF-8

FIGURE 7.2 Examples of manifest files for single-end and paired-end FASTQ files.

```
qiime tools import \
--type 'SampleData[PairedEndSequencesWithQuality]' \
--input-format PairedEndFastqManifestPhred33 \
--input-path data \
--output-path artifacts/xxxx.qza
```

The artifacts created by the above commands are demultiplexed and they are ready for processing by QIIME2.

7.3.1.2 Metadata

In the above, we discussed importing the raw sequence data into QIIME2 artifacts. But we have also mentioned that a sample metadata describing the study is required for the analysis. A metadata file can also be created by the user and will be used later in the analysis to show the sample grouping. A metadata file is a tab-separated value (TSV) file containing sample and study design information in columns and the first column must be the sample identifiers for the IDs of the sample in the study. The top line is for the column names. The ID column name must be one of those: id, sampleid, sample id, sample-id, featured, feature id, or feature-id. The IDs listed in the metadata files should be unique and not empty. The ID column is not optional and the metadata file must contain at least one ID. The ID column can be followed by additional columns defining metadata associated with each sample or feature such as groups, treatment, factor, and barcode sequence. The column name should not be one of the reserved words and the type of the data contained in the metadata file is either categorical or numeric. The sample metadata is created by the user manually. However, if the data is downloaded from the NCBI SRA database, the metadata can also be downloaded and modified to meet our goal. This will be discussed below in the worked example.

7.3.2 Demultiplexing

Above, we discussed the EMP and non-EMP-multiplexed FASTQ files and how they can be imported into QIIME2 artifacts. Since the reads are multiplexed, demultiplexing must be carried out before reads processing and analysis. QIIME2 uses “demux” and “cutadapt” plugins for demultiplexing EMP-multiplexed reads and non-EMP-multiplexed reads, respectively.

```

qiime demux emp-single          # EMP-multiplexed single-end reads
qiime demux emp-paired          # EMP-multiplexed paired-end reads
qiime cutadapt demux-single     # non-EMP-multiplexed single-end reads
qiime cutadapt demux-paired      # non-EMP-multiplexed paired-end reads

```

The following is a format for demultiplexing EMP-multiplexing single-end reads. Notice that the input for the command is an artifact that is created by importing a FASTQ file as discussed above.

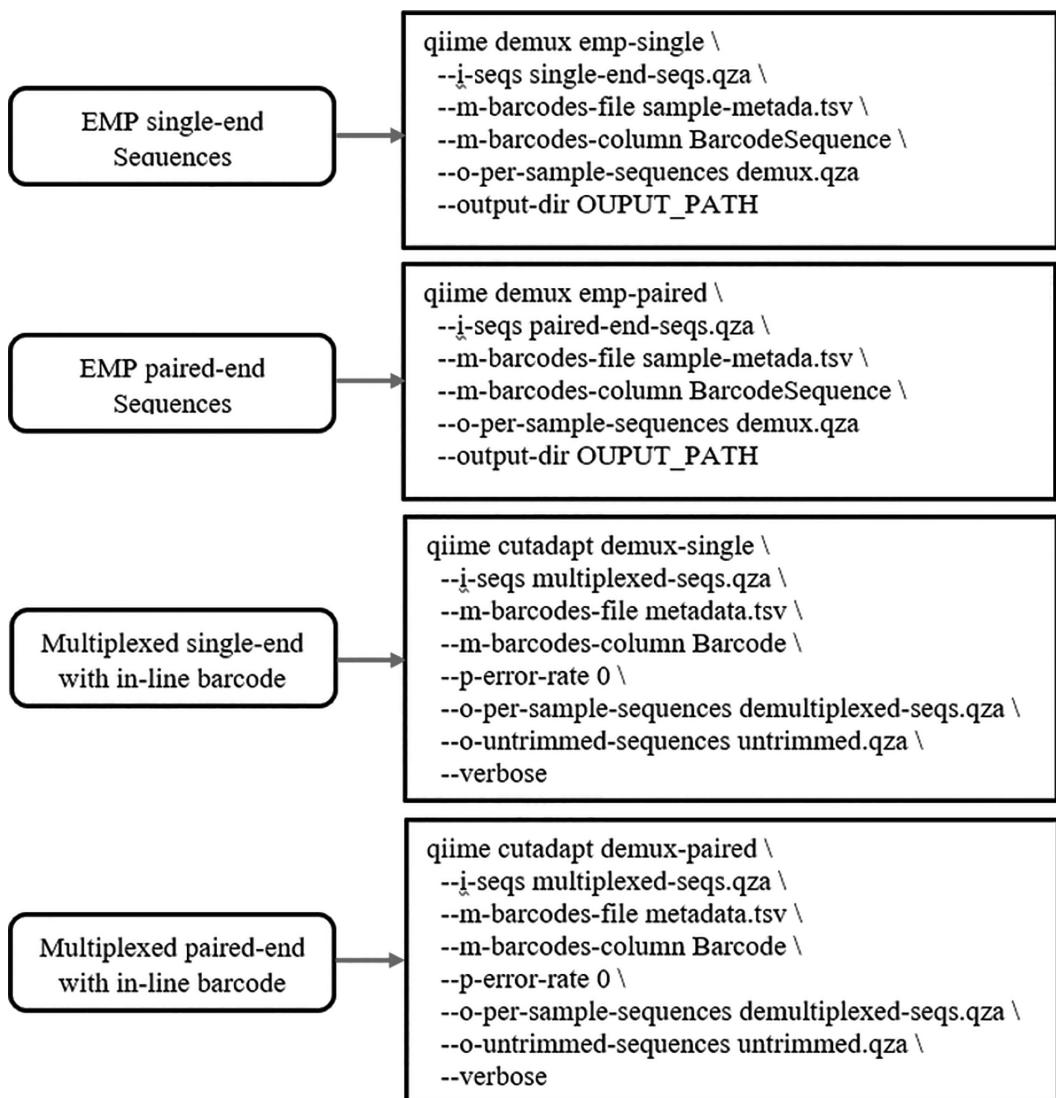


FIGURE 7.3 Demultiplexing plugin for different multiplexed data types.

```
qiime demux emp-single \
--i-seqs artifacts/multiplexed-emp-single-end.qza \
--m-barcodes-file data/sample-metadata.tsv \
--m-barcodes-column barcode-sequence \
--o-per-sample-sequences artifacts/demux-single-end.qza \
--o-error-correction-details artifacts/demux-details.qza
```

It is also required to provide the sample metadata file as an input “m-barcodes-file” and the column which includes the barcode sequence in the metadata to be specified. The output is two artifacts: an artifact for the demultiplexed reads and an artifact for the demultiplexing details.

Figure 7.3 shows the commands and usages for demultiplexing both file formats.

Once the raw data is imported into QIIME2 artifact and the multiplexed reads were demultiplexed, all types of raw data will be preprocessed and analyzed in the same way. Therefore, we will discuss the remaining steps of the analysis with QIIME2 through a worked example.

7.3.3 Downloading and Preparing the Example Data

As an example, we will download sequence raw data from the NCBI SRA database. The data is for amplicon-based 16S rRNA gene sequences obtained from NGS for a study to examine the effect of a yoga-based intervention against a low-FODMAP diet on patients with irritable bowel syndrome. FODMAP is an acronym for Fermentable Oligosaccharides, Disaccharides, Monosaccharides, and Polyols, which are short-chain carbohydrates and poorly absorbed in the small intestine. The metagenomic 16S rRNA gene data sequenced from fecal samples are available for 86 patients, with irritable bowel syndrome, grouped into (i) patients who received yoga sessions and (ii) patients who received low-FODMAP diet. The NCBI BioProject accession for this study is PRJEB24421. We will download the FASTQ files from the NCBI SRA database and then we will follow through the QIIME2 pipeline to analyze these data step by step. The data are for demultiplexed paired-end sequences: two FASTQ files (forward and reverse) for each sample.

7.3.3.1 Downloading the Raw Data

To download the files of all experiments, we need to obtain the run accessions of the experiments in the BioProject. To keep files organized, we will create a directory to store the raw data of this project. Open the Linux terminal and create a directory with the BioProject accession, and inside that directory, create a subdirectory with the name “data” as follows:

```
mkdir PRJEB24421
cd PRJEB24421
mkdir data
```

In the next step, we will save the run accessions of the BioProject in a text file in the “data” subdirectory. To do that, you can open the NCBI SRA database and search for the

BioProject by the above accession number or simply copy and paste the following URL on the Internet browser:

<https://www.ncbi.nlm.nih.gov/sra/?term=PRJEB24421>

Then, use “Send to” dropdown menu to download the runinfo text file. After downloading the text file, open the file in Excel, delete all columns except the column with the run accessions and remove the column name as well, and save the file as “runids.txt” in the “data” subdirectory.

Instead of the above, you can also use the following EDirect script, which extracts the run accessions and stores them in a file named “runids.txt” in “data” subdirectory (you should have the NCBI Entrez Direct installed):

```
esearch -db sra -query 'PRJEB24421[bioproject]' \
| efetch -format runinfo \
| cut -f1 -d, > data/runids.txt
sed -i '/^$/d' data/runids.txt
sed -i '/^Run/d' data/runids.txt
```

Check to see if the file has been saved successfully by using “ls data/” command or you can display the file content by using “vim data/runids.txt” command.

After saving the text file with the 86 run accessions in the “data/runids.txt” file, you can then download the raw FASTQ files from the NCBI SRA database either by saving the following script in a bash file “download.sh” and then run it as “bash download.sh” or you can just enter the script on the terminal command-line prompt, while you are in the project directory:

```
while read f;
do
    fasterq-dump --progress --outdir data "$f"
done < data/runids.txt
```

You will see the downloading progress. The files require only 771.29MB of storage space. The 172 FASTQ files will be downloaded in the “data” subdirectory, two files for each sample. When the files have been downloaded successfully, you can check the content of the “data” subdirectory and count the number of the FASTQ files using the following command:

```
ls data/*.fastq | wc -l
```

The number of files should be 172. If it is not, you may need to run the download script again.

7.3.3.2 Creating the Sample Metadata File

Open the NCBI SRA using the above URL. Then, open Run Selector from “Send to” dropdown menu. All runs will be displayed on the Run Selector. Click “Metadata” button

FIGURE 7.4 Downloading metadata from SRA database using Run Selector.

sample-id	Group	sampleName	time
ERR2247316	FODMAP	10.Dn8	t0
ERR2247320	Yoga	104.Dj1	t1
ERR2247323	FODMAP	15.Dn4	t0
ERR2247325	FODMAP	17.Dn23	t0
ERR2247326	Yoga	18.Dj2	t0
ERR2247327	FODMAP	19.Dn5	t0
ERR2247331	FODMAP	23.Dj7	t0
ERR2247333	Yoga	25.Dn40	t0
ERR2247335	FODMAP	27.Dn26	t0
ERR2247337	FODMAP	30.Dn27	t0
ERR2247338	FODMAP	31.Dj1	t0
ERR2247339	FODMAP	32.Dj3	t0
ERR2247342	Yoga	37.Dj16	t0

FIGURE 7.5 The sample metadata text file “sample-metadata.tsv”.

(Figure 7.4) to download the metadata text file. Open the file in a spreadsheet, remove all columns except “run”, “group”, and “time”. Then, rename “run” to “SampleID”, insert a data type row as shown in Figure 7.5, and save the file as a TSV file with the name: “data/sample-metadata.tsv”.

If you have EDirect installed on your computer, you can search the NCBI database and retrieve metadata from it. EDirect is a collection of Linux-based command-line E-Utilities programs. The instructions for installing EDirect programs are available at the NCBI

website at “<https://dataguide.nlm.nih.gov/edirect/install.html>”. You can use the following script to use EDirect to create a metadata file. The script searches the NCBI SRA database for the BioProject “PRJEB24421” and then it retrieves the sample metadata and stores them in a TSV file “sample-metadata.tsv”.

```
esearch -db sra -query 'PRJEB24421[bioproject]' \
| efetch -format runinfo \
| tr -s ',' '\t' > sample-metadata.tsv
```

Then, you can edit the file as above.

7.3.3.3 Importing Microbiome Yoga Data

Our example raw data is non-Casava 1.8 demultiplexed reads. To import the FASTQ files into QIIME2 artifact, we need a manifest file listing the file names and their absolute path as described above. We can create the manifest file manually or we can use the following bash script. Before running the script, change to the “data” directory “cd data”, where the FASTQ files are found.

```
#Creating a manifest file
#####
#a- make file name and absolute path
find "$PWD"/*.fastq -type f -printf '%f %h/%f\n' > tmp.txt
#b- remove _1/2.fastq
awk '{ gsub(/_[12].fastq/,",", $1); print } ' tmp.txt > tmp2.txt
#remove space
cat tmp2.txt | sed -r 's/^\s+//g' > tmp3.txt
n=$(ls -l *1.fastq|wc -l)
#create a direction column
seq $n | sed "c forward\nreverse" > tmp4.txt
#add direction column
paste tmp3.txt tmp4.txt | column -s $' ' -t > tmp5.txt
#replace space with comma
sed -e 's/^\s+/,/g' tmp5.txt > manifest.txt
#add column names
sed -i '1s/^/sample-id,absolute-filepath,direction\n/' manifest.
txt
rm tmp*.txt
```

The “manifest.txt” file will be created in “data” directory, and it looks as shown in Figure 7.6.

After running the above script, you can display the file content using the text editor of your choice. Then, move back to the project main directory using “cd ..”.

The next step is to import the FASTQ files into a QIIME2 artifact. To keep files organized, you can create a new subdirectory “input” for the artifact files.

```
mkdir inputs
```

```
sample-id,absolute-filepath,direction
ERR2247316,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247316_1.fastq,forward
ERR2247316,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247316_2.fastq,reverse
ERR2247317,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247317_1.fastq,forward
ERR2247317,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247317_2.fastq,reverse
ERR2247318,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247318_1.fastq,forward
ERR2247318,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247318_2.fastq,reverse
ERR2247319,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247319_1.fastq,forward
ERR2247319,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247319_2.fastq,reverse
ERR2247320,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247320_1.fastq,forward
ERR2247320,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247320_2.fastq,reverse
ERR2247321,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247321_1.fastq,forward
ERR2247321,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247321_2.fastq,reverse
ERR2247322,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247322_1.fastq,forward
ERR2247322,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247322_2.fastq,reverse
ERR2247323,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247323_1.fastq,forward
ERR2247323,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247323_2.fastq,reverse
ERR2247324,/home/hamiddafa/qiime2/PRJEB24421/fastq/ERR2247324_1.fastq,forward
```

FIGURE 7.6 Manifest file for the example FASTQ files.

The following script uses the “qiime tools import” to import the 172 FASTQ files onto a single QIIME2 artifact “demux-yoga.qza” in the “inputs” directory:

```
qiime tools import \
--type 'SampleData[PairedEndSequencesWithQuality]' \
--input-format PairedEndFastqManifestPhred33 \
--input-path data/manifest.txt \
--output-path inputs/demux-yoga.qza
```

We will use this artifact “demux-yoga.qza” in the next step.

7.3.4 Raw Data Preprocessing

All the raw FASTQ files are contained in a single QIIME2 artifact. The next step would be data preprocessing, which includes quality control and denoising.

7.3.4.1 Quality Assessment and Quality Control

QIIME2 has plugins for quality assessment and quality control. We can use “q2-demux summarize” to assess the quality of the raw data in the artifact. The output of that command is a visualization file containing the quality assessment report that can be viewed on an Internet browser. To keep the files organized, we can create a subdirectory to the visualization file called it “viz”.

```
mkdir viz
```

Then, we can run the following script to save the report visualization file in the “viz” subdirectory:

```
qiime demux summarize \
--i-data inputs/demux-yoga.qza \
--o-visualization viz/demux-yoga-qc.qzv
```

We can view the visualization file on the Internet browser using “q2-tools view” as follows:

```
qiime tools view viz/demux-yoga-qc.qzv
```

The report displayed on the Internet browser will have two tabs: Overview and Interactive Quality Plot. The Overview report tab shows Demultiplexed sequence count summary, forward and reverse reads frequency histogram, and Per-sample sequence counts. The summary count section shows minimum, median, mean, maximum, and total number of reads across the samples for forward reads and reverse reads (this is only in paired end). Figure 7.7 shows that the summary statistics for forward reads and reverse reads are the same. The minimum number of reads in a sample is 7702 and the maximum number of reads is 95075. The median and mean of reads across all samples are 25588.0 and 30018.709302, respectively. The total number of forward reads in all samples is 2581609 and is the same for reverse reads. The histogram shows the distribution of the number of reads versus number of samples. The histograms can be downloaded as pdf files. The Per-sample sequence counts table, below the histogram, lists sample ID and number of forward reads and reverse reads in each sample. The Interactive Quality Plot section shows plots for the per base Phred quality score across reads for both forward and reverse reads (Figure 7.8). These plots are similar to the QC plots discussed in Chapter 1. They will help us to determine if the sequences need trimming or filtering. If you hover the mouse over a specific position on the plot, the table below the plot will update itself to display the seven-number summary statistics for that base position. This table corresponds to what is visually represented by the box plot at that position.

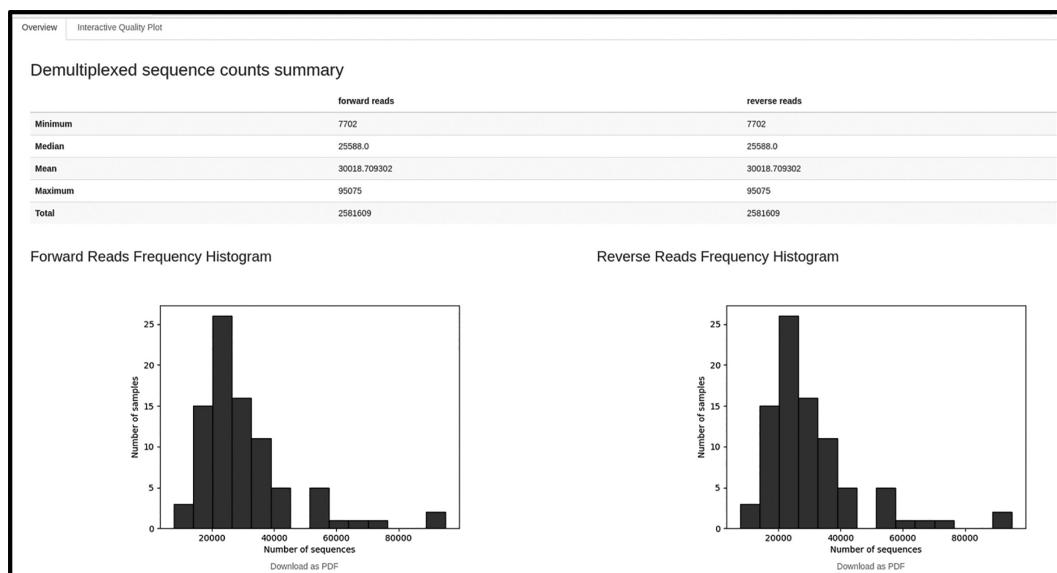


FIGURE 7.7 Sequence count summary and histograms of reads of yoga data.

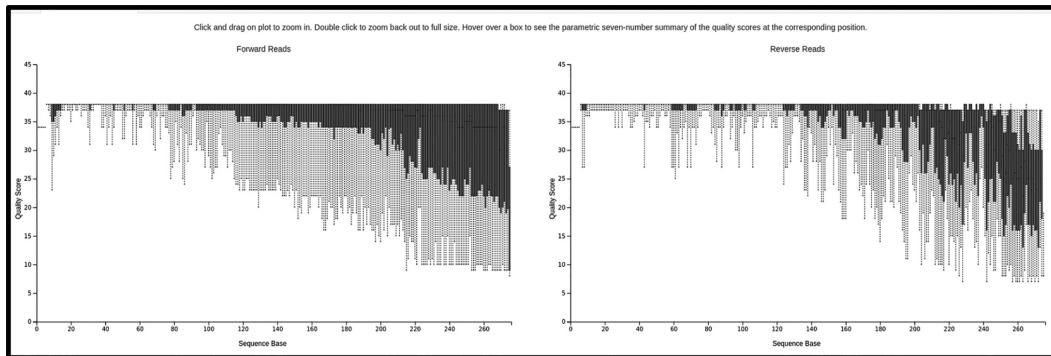


FIGURE 7.8 Per base quality plots of the yoga data.

You can notice that the overall quality scores of the reads are high but there are also some reads with quality score less than 20 (99% accuracy) toward the end of the reads. We can trim the low-quality bases from the end of the reads. The demultiplexed sequence length summary at the bottom of the Interactive Quality Plot tab shows that the reads have equal length (275 base). This table will help us to determine if we need to make the length of reads equal or not.

If you decide to filter out the reads with poor quality scores, you can use the “quality-filter” plugin with “q-score” method. However, this can be done for single-end reads. For paired-end reads, you can join forward and reverse reads and then run “quality-filter q-score” on the merged reads. This will be discussed later with clustering. However, denoising methods also have their way to filter low-quality reads as we will see soon. But, if your data is single-end reads, you can use “quality-filter q-score” to remove low-quality reads using the following script:

```
qiime quality-filter q-score \
--i-demux demux.qza \
--p-min-quality 20 \
--p-quality-window 5 \
--p-min-length-fraction 0.8 \
--p-max-ambiguous 0 \
--o-filtered-sequences demux-filtered.qza \
--o-filter-stats demux-filter-stats.qza
```

The default settings are “--p-min-quality 4”, “--p-quality-window 3”, “--p-min-length-fraction 0.75”, and “--p-max-ambiguous 0”, if those parameters are not included in the above. For more information about these parameters, use “qiime quality-filter q-score --help”. We will discuss this in more detail with clustering.

If there are PCR primer sequences or any other non-biological sequences, you can remove them using “cutadapt” plugin. Once more, this is not applicable to our yoga data but, just in case, if you had sequences with primers at this stage of the analysis, it would

be the right moment to remove them using “cutadapt” plugin with “trim-single” or “trim-paired” for single-end or paired-end reads, respectively.

```
qiime cutadapt trim-single \
--i-demultiplexed-sequences demux.qza \
--p-front GTGCCAGCMGCCGCGTAA \
--p-error-rate 0 \
--o-trimmed-sequences trimmed-demux.qza \
--verbose
```

About our yoga data, now we have a clear view about its quality after we have assessed it using “demux summarize”. Since the data is paired-end reads, we will carry out the quality control later with clustering and denoising.

7.3.4.2 Clustering and Denoising

After the above preprocessing, the next step in the data analysis is to create features by either clustering or denoising as discussed above. QIIME2 supports de novo, closed-reference, and open-reference clustering using VSEARCH plugin and denoising with DADA2 and deblur. Either clustering or denoising is performed to create feature tables and representative sequences. Denoising (with DADA2 or deblur) attempts to remove the noises generated from errors. The features generated by DADA2 and deblur are also called amplicon sequence variant (ASVs). Whatever you choose to continue with clustering or with denoising, it is your sole choice. These techniques were discussed above in detail. In the following, we will show you how to perform clustering and denoising with QIIME2.

7.3.4.2.1 Clustering

If your plan is to cluster reads into OTUs without denoising, QIIME2 provides “q2-vsearch” plugin to do just that. This plugin has methods for the three types of clustering: de novo, closed-reference, and open-reference. The “q2-vsearch” plugin can also perform quality control; therefore, before running clustering, you may need to do some preprocessing to the data. The paired-end reads must be merged before processing. In the following, we will walk you through the steps of clustering to the point of generating feature tables and OTU representative sequences.

7.3.4.2.1.1 Merging Paired-End Reads

If the data is paired-end reads, the forward and reverse reads must be merged before clustering. The merging is achieved with “join-pairs” method of “q2-vsearch” plugin.

The artifact “demux-yoga.qza” of our example data is in the “inputs” directory. Since the reads are paired end, we can merge them before clustering. The following script takes the “demux-yoga.qza” artifact as an input, joins the forward and reverse reads, and creates a new artifact for the merged reads “demux-yoga-merged.qza”:

```
qiime vsearch join-pairs \
--i-demultiplexed-seqs inputs/demux-yoga.qza \
```

```
--p-allowmergestagger \
--o-joined-sequences inputs/demux-yoga-merged.qza
```

For checking the merging, run “demux summarize” to create a visualization file for the summary and quality of the merged reads as discussed above.

```
qiime demux summarize \
--i-data inputs/demux-yoga-merged.qza \
--o-visualization viz/demux-yoga-merged-qc.qzv
qiime tools view viz/demux-yoga-merged-qc.qzv
```

If you open Interactive quality plot on the summary report, you will notice that the forward and reverse reads have been joined as shown in Figure 7.9.

If, at this point, you notice that the forward reads (for single-end data) or merged reads (for paired-end data) require filtering, you can then perform that with “quality-filter q-score” as follows:

```
qiime quality-filter q-score \
--i-demux inputs/demux-yoga-merged.qza \
--o-filtered-sequences inputs/demux-yoga-merged-filter.qza \
--o-filter-stats inputs/demux-yoga-merged-filter-stats.qza
```

7.3.4.2.1.2 Sequence Dereplication

Before clustering, the reads must be dereplicated to reduce the redundancy of the reads based on similarity. The dereplication is carried out with “dereplicate-sequences” method

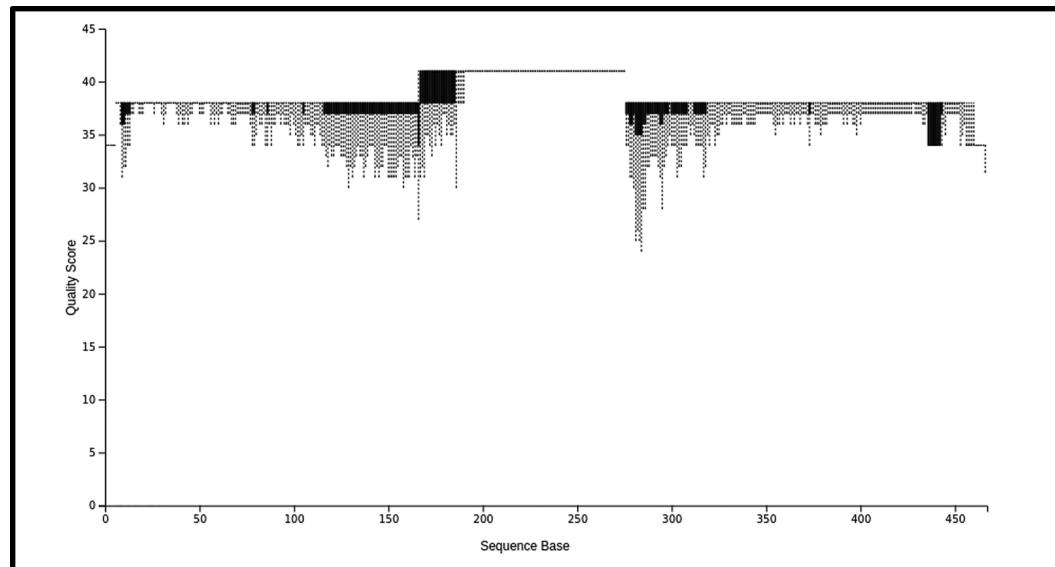


FIGURE 7.9 Per base quality plots of the merged yoga data.

of “q2-vsearch” plugin. The input is the last preprocessed data artifact “demux-yoga-merged.qza”.

```
qiime vsearch dereplicate-sequences \
--i-sequences inputs/demux-yoga-merged.qza \
--o-dereplicated-table inputs/derep-yoga-table.qza \
--o-dereplicated-sequences inputs/derep-yoga-seqs.qza
```

The outputs from the “dereplicate-sequences” command are two artifacts: (i) feature table containing the OTU features with their observed abundances (frequencies) for each of the samples of the study and (ii) feature data in which each feature identifier is mapped to a feature.

7.3.4.2.1.3 Clustering Methods

Clustering then follows the dereplication. Both feature table and feature data artifacts generated with the dereplication are required as inputs for clustering. We will use QIIME2 to perform the three types of clustering (de novo, closed-reference, and open-reference clustering).

7.3.4.2.1.3.1 De Novo Clustering

The de novo clustering does not require a database but it uses sequence similarity to cluster features into groups. The threshold for similarity can be set to 0.99% (only reads similar to centroid sequence with identity of 99% are allowed to join the cluster). QIIME2 uses “cluster-features-de-novo” method of “q2-vsearch” plugin to perform the de novo clustering. The input artifacts are the feature table and feature data artifacts generated by the dereplication in the previous step. To keep the clustering in a separate directory, we will create the “denovo” subdirectory.

```
mkdir denovo
qiime vsearch cluster-features-de-novo \
--i-table inputs/derep-yoga-table.qza \
--i-sequences inputs/derep-yoga-seqs.qza \
--p-perc-identity 0.99 \
--o-clustered-table denovo/table-yoga-denovo.qza \
--o-clustered-sequences denovo/rep-seqs-yoga-denovo.qza
```

The outputs are two artifacts: a feature table for the OTUs and feature data that contains the centroid sequences defining each OTU cluster. De novo clustering usually consumes more computational resources compared to the other two methods.

7.3.4.2.1.3.2 Closed-Reference Clustering

Closed-reference clustering requires a curated database for the 16S rRNA gene sequences as reference sequences. Only the representative sequences that have matches on the database are clustered, while the ones that do not have matches will be discarded. Examples of widely used databases include Greengenes (16S rRNA) at “<https://greengenes.secondgenome.org>”.

com”, Silva (16S/18S rRNA) at <https://www.arb-silva.de>, and UNITE (fungal ITS) at “<https://unite.ut.ee/>”. The database must be downloaded and then imported in QIIME2 as artifact before being used for clustering. For example, we will download the latest release of curated OTUs from GreenGenes database:

```
wget ftp://greengenes.microbio.me/greengenes_release/gg_13_5/
gg_13_8_ottus.tar.gz
tar vxf gg_13_8_ottus.tar.gz
rm gg_13_8_ottus.tar.gz
```

Make sure that the URL is a single line with no white space. You can visit the website to download the latest release.

The files will be extracted into a directory (gg_13_8_ottus). Display the contents of this directory and its subdirectories using “ls” Linux command. You will find four subdirectories: “ottus” (for reference OTUs), “rep_set” (for the reference representative sequences), “rep_set_aligned” (for aligned representative sequences), “taxonomy” (for taxonomy files), and “trees” (for phylogenetic trees). The files in these directories contain data at different identities (e.g., 99%, 97%, and 94%). Keep this database as we will use it for other applications.

To use the reference database for clustering, you need to import the file of the database representative sequences (FASTA file). You need to choose at which identity you wish to perform clustering. Assume that you want to cluster your sample sequences at 97% identity, then you can import “rep_set/97_ottus.fasta” onto QIIME2 as artifact using “tools import”. To keep the files organized, we will create the subdirectory “closed_ref_cl_97” for closed-reference clustering files.

```
mkdir closed_ref_cl_97
```

Then, import the database representative sequences into QIIME2 artifact.

```
qiime tools import \
--type 'FeatureData[Sequence]' \
--input-path gg_13_8_ottus/rep_set/97_ottus.fasta \
--output-path inputs/97_ottus-GG_db.qza
```

Then, you can use the “cluster-features-closed-reference” method of the “q2-vsearch” plugin to perform the closed-reference clustering on the features generated in the dereplication steps. The input artifacts are: dereplicated feature table “derep-yoga-table.qza”, dereplicated representative sequences “derep-yoga-seqs.qza”, and the reference representative sequences from the database “97_ottus-GG_db.qza”.

```
qiime vsearch cluster-features-closed-reference \
--i-table inputs/derep-yoga-table.qza \
--i-sequences inputs/derep-yoga-seqs.qza \
```

```
--i-reference-sequences inputs/97_otus-GG_db.qza \
--p-perc-identity 0.97 \
--o-clustered-table closed_ref_cl_97/table-yoga-closed_cl.qza \
--o-clustered-sequences closed_ref_cl_97/rep-seqs-yoga-close_
cl.qza \
--o-unmatched-sequences closed_ref_cl_97/
unmatched-yoga-close_cl.qza
```

The above script outputs three artifacts: A feature table, clustered sequences (the sequences defining the features in the feature table), and unmatched sequences (the sequences that didn't match reference sequences at 97% identity). The unmatched sequences will be completely ignored.

7.3.4.2.1.3.3 Open-Reference Clustering

The open-reference clustering is hybrid of the above two clustering methods. First, it uses reference sequences for clustering the matched sequences and then it performs de novo clustering on the unmatched sequences. The open-reference clustering is performed with “cluster-features-open-reference” method. The input and output artifacts are the same as that of the closed-reference clustering except that there are no unmatched sequences; instead, there is an artifact for the new reference sequences used as an input in addition to the sequences clustered as part of the internal de novo clustering step. We will create the new subdirectory “open_ref_cl_97” for files of the open-reference clustering.

```
mkdir open_ref_cl_97
qiime vsearch cluster-features-open-reference \
--i-table inputs/derep-yoga-table.qza \
--i-sequences inputs/derep-yoga-seqs.qza \
--i-reference-sequences inputs/97_otus-GG_db.qza \
--p-perc-identity 0.97 \
--o-clustered-table open_ref_cl_97/table-yoga-open_cl.qza.qza \
--o-clustered-sequences open_ref_cl_97/rep-seqs-yoga-open_cl.qza \
--o-new-reference-sequences open_ref_cl_97/
new-ref-seqs-open_cl.qza
```

The three clustering methods use dereplicated feature table and representative sequences and produce a final feature table and OTU representative sequences to be used in the downstream analysis for phylogeny, diversity analysis, assignment of taxonomic group, and differential taxonomic analysis.

7.3.4.2.2 Denoising

Like clustering, denoising also produces a feature table and representative sequences. However, denoising attempts to remove errors and to provide more accurate results.

There are two denoising methods available in QIIME2: DADA2 and deblur. Both methods output feature tables containing feature abundances and ASVs. Moreover, they also

perform quality filtering and chimera removal. You may not need to perform any quality control prior using any of these two methods. Only for deblur denoising, you may need to merge paired-end reads as we did for the clustering.

7.3.4.2.2.1 Denoising with DADA2

The “q2-dada2” plugin is used to denoise single-end and paired-end reads (no need for read merging). Use “denoise-single” method for the single-end reads and “denoise-paired” method for the paired-end reads. You can always use “--help” with any of the methods to display the usage and options. DADA2 methods denoise sequences, derePLICATE them, and filter chimeras.

```
qiime dada2 denoise-single --help
qiime dada2 denoise-paired --help
```

Now, we can use “q2-dada2” to denoise the yoga data that we imported and saved as “demux-yoga.qza”. We will use “denoise-paired” method. To keep the files organized, we will create the “dada2” subdirectory for DADA2 denoising files.

```
mkdir dada2
qiime dada2 denoise-paired \
--i-demultiplexed-seqs inputs/demux-yoga.qza \
--p-trim-left-f 0 \
--p-trim-left-r 0 \
--p-trunc-len-f 250 \
--p-trunc-len-r 250 \
--p-n-threads 4 \
--o-representative-sequences dada2/rep-seqs_yoga_dada2.qza \
--o-table dada2/table_yoga_dada2.qza \
--o-denoising-stats dada2/stats_yoga_dada2.qza
```

The parameters “--p-trim-left-f”, “--p-trim-left-r”, “--p-trunc-len-f”, and “--p-trunc-len-r” are optional, and they are used to trim and truncate the forward and reverse sequences, respectively, to improve the quality of the reads if required. If you use “--p-trunc-len-f 0” and “--p-trunc-len-r 0”, the truncation will be disabled. The parameter “--p-n-threads” specifies the number of threads used for denoising. If the “denoise-single” method is used with paired-end reads instead of “denoise-paired”, only forward reads will be used as input while the reverse reads will be ignored.

We set “--p-trunc-len-f 250” and “--p-trunc-len-r 250” to truncate the forward and reverse reads to 250 bases. However, we did not trim the left ends of the reads because they do not need trimming. Like clustering, DADA2 feature table and representative sequences artifacts are used for the downstream analysis for phylogeny, diversity analysis, taxonomy assignment, etc. Moreover, the DADA2 stats summary artifact contains useful information regarding the filtering and denoising. Users can use “q2-metadata tabulate” to generate a visualization file that can be displayed on the Internet browser with “tools view” as follows:

```
qiime metadata tabulate \
  --m-input-file dada2/stats_yoga_dada2.qza \
  --o-visualization dada2/stats_yoga_dada2.qzv
qiime tools view dada2/stats_yoga_dada2.qzv
```

The DADA2 stats summary table is a TSV. It shows the ID, number of input reads, number of reads filtered, percentage of reads passed the filter, number of denoised reads, number of merged reads, percentage of reads merged, number of non-chimeric reads, and percentage of non-chimeric reads for each sample in the study.

7.3.4.2.2.2 Denoising with Deblur

Deblur plugin denoises single-end reads and merged paired-end read. Paired-end sequences must be merged using “vsearch join-pairs”, as shown above, before being denoised by deblur.

```
mkdir deblur
qiime vsearch join-pairs \
  --i-demultiplexed-seqs inputs/demux-yoga.qza \
  --p-allowmergestagger \
  --o-joined-sequences deblur/demux-yoga-merged.qza
```

It is recommended to run quality assessment and quality control by filtering low-quality reads using “quality-filter q-score” before denoising with deblur.

```
qiime quality-filter q-score \
  --i-demux inputs/demux-yoga-merged.qza \
  --o-filtered-sequences deblur/demux-yoga-merged-filtered.qza \
  --o-filter-stats deblur/demux-yoga-merged-filtered-stat.qza
```

Denoising with deblur is carried out with “q2-deblur” plugin, which has two methods: “denoise-16S” for denoising 16S rRNA gene sequences and “denoise-other” for denoising other types of targeted gene sequences. When you use “denoise-16S”, deblur will perform an initial positive filtering step by discarding any reads that do not have a minimum 60% identity similarity to database sequences from the 85% GreenGenes.

You can use “deblur denoise-16S” method, which has “--p-trim-length” parameter that can be set to truncate the reads at specific position for quality filtering or you can set that parameter to -1 to disable trimming. The following script will perform deblur denoising:

```
mkdir deblur
qiime deblur denoise-16S \
  --i-demultiplexed-seqs deblur/demux-yoga-merged.qza \
  --p-trim-length -1 \
  --p-jobs-to-start 4 \
  --p-sample-stats \
  --o-representative-sequences deblur/rep-seqs_yoga_deblur.qza \
```

```
--o-table deblur/table_yoga_deblur.qza \
--o-stats deblur/stats_yoga_deblur.qza
```

The “--p-jobs-to-start” is an optional parameter and you can set it to the number of jobs to run in parallel. To learn about more options, use “qiime deblur denoise-16S --help”.

The feature table and representative sequences artifacts will be used in the downstream analysis.

The deblur stats summary artifact contains useful information about the filtering and denoising. We can use the “deblur visualize-stats” plugin to generate a visualization file (Figure 7.10).

```
qiime deblur visualize-stats \
--i-deblur-stats deblur/stats_yoga_deblur.qza \
--o-visualization deblur/stats_yoga_deblur.qzv
qiime tools view deblur/stats_yoga_deblur.qzv
```

Both clustering and denoising (with DADA2 or deblur) generate feature table and representative sequences artifacts that can be used in the following analysis steps. You may need to visualize these feature table and the sequence data artifacts. The “q2-feature-table” plugin is used just for that. The “summarize” and “tabulate-seqs” methods are used to create a visualization file for the feature table and representative sequences, respectively. As examples, the following script creates visualization files for the feature tables produced by the de novo clustering and DADA2 denoising, respectively. The “summarize” method of the “feature-table” plugin takes a feature table artifact and the sample metadata file as input and it outputs a visualization file. We created the sample metadata file “sample-metadata.tsv” earlier and we saved it in the “data” subdirectory.

	sample-id	reads-raw	fraction-artifact-with-minsize	fraction-artifact	fraction-missed-reference	unique-reads-derep	unique-reads-derep	unique-reads-deblur	unique-reads-deblur	unique-reads-hit-artifact	reads-hit-artifact	unique-reads-chimeric	reads-chimeric	unique-reads-hit-reference	reads-hit-reference	unique-reads-missed-reference	reads-missed-reference
0	ERR2247376	7936	0.907888	0.0	0.0	128	731	51	202	0	0	9	13	26	153	0	0
1	ERR2247353	10766	0.902749	0.0	0.0	171	1047	66	241	0	0	8	9	32	173	0	0
2	ERR2247322	17320	0.902252	0.0	0.0	306	1693	118	467	0	0	10	21	55	320	0	0
3	ERR2247356	13696	0.900190	0.0	0.0	215	1367	82	348	0	0	9	14	50	288	0	0
4	ERR2247341	22756	0.894577	0.0	0.0	370	2399	144	649	0	0	21	36	62	467	0	0
5	ERR2247386	10551	0.890816	0.0	0.0	212	1152	86	288	0	0	10	13	43	198	0	0
6	ERR2247369	43054	0.886747	0.0	0.0	461	4876	148	1088	0	0	19	52	66	880	0	0
7	ERR2247382	18494	0.882611	0.0	0.0	262	2171	107	595	0	0	19	37	46	477	0	0

FIGURE 7.10 Deblur denoising stats summary.

```

qiime feature-table summarize \
--i-table denovo/table-yoga-denovo.qza \
--m-sample-metadata-file data/sample-metadata.tsv \
--o-visualization denovo/table-yoga-denovo.qzv
qiime tools view denovo/table-yoga-denovo.qzv

qiime feature-table summarize \
--i-table dada2/table_yoga_dada2.qza \
--m-sample-metadata-file data/sample-metadata.tsv \
--o-visualization dada2/table_yoga_dada2.qzv
qiime tools view dada2/table_yoga_dada2.qzv

```

The html report of the feature table visualization, as displayed on the Internet browser (Figure 7.11), has three tabs: Overview, Interactive Sample Details, and Feature Details tab. Overview tab contains Table summary, Frequency per sample with a histogram, and Frequency per feature with a histogram. Interactive Sample Detail tab (Figure 7.12) displays a plot, a table containing sample IDs and feature counts, and Plot controls to control the plot interactively by changing Metadata Category using the dropdown list and sampling depth using Sampling Depth sliding bar. The metadata category list is obtained from the sample metadata file which describes the sample groups. Feature Details tab (Figure 7.13) shows the list of the features found in the samples. The table includes feature id, feature frequency (abundance), and number of samples in which that feature is observed.

The “feature-table summarize” methods are used to visualize any feature tables generated by any of the clustering or denoising methods.

We can also use “tabulate-seqs” to create a visualization file for the representative sequence artifact generated by any of the clustering or denoising methods.

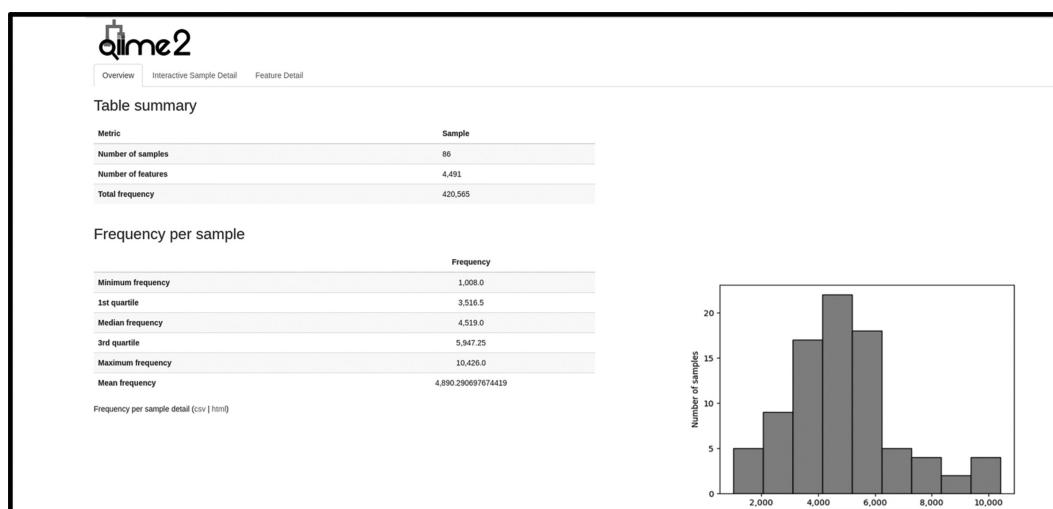


FIGURE 7.11 The Stats summary of the feature table.

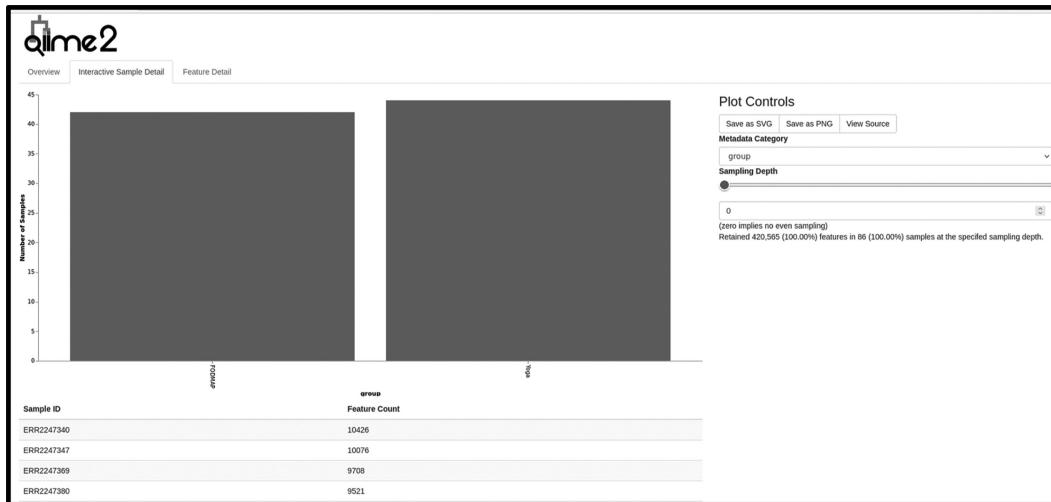


FIGURE 7.12 Interactive sample detail of the feature table.

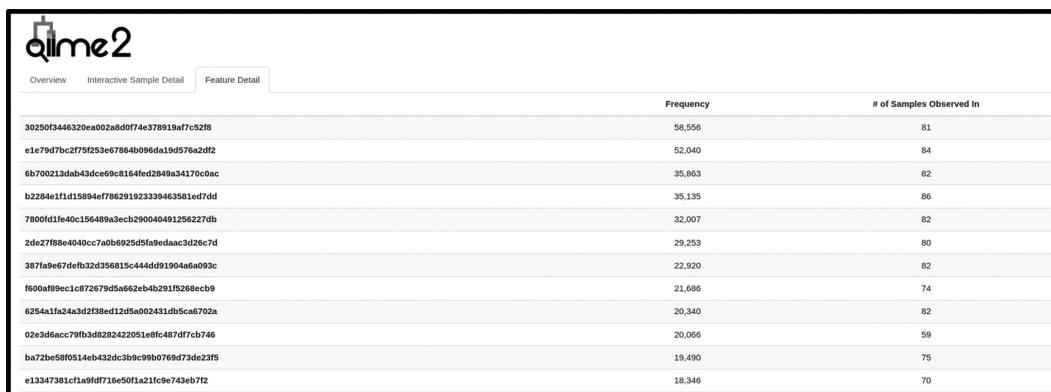


FIGURE 7.13 Feature detail of feature table.

The following script creates a visualization file for the representative sequence artifact produced with deblur denoising:

```
qiime feature-table tabulate-seqs \
--i-data deblur/rep-seqs_yoga_deblur.qza \
--o-visualization deblur/rep-seqs_yoga_deblur.qzv
qiime tools view deblur/rep-seqs_yoga_deblur.qzv
```

The tabular report of the representative sequences (Figure 7.14), as displayed on the Internet browser, contains Sequence Length Statistics, Seven-Number Summary of Sequence Lengths, and Sequence Table that includes ID, sequence length, and sequence for each feature. Moreover, each feature sequence is linked to a BLAST report showing the sequences with significant alignments to the feature sequence.

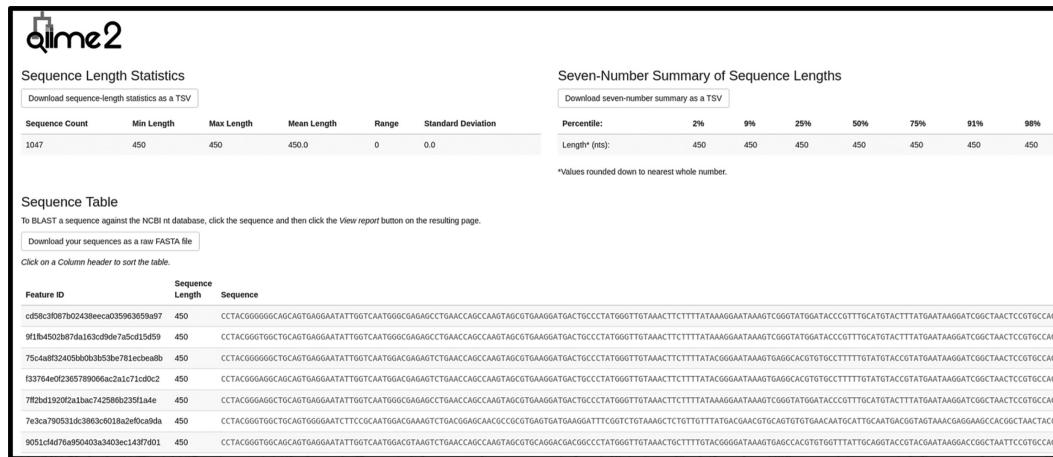


FIGURE 7.14 The tabular report of the amplicon sequence variants (ASVs).

After you study the feature table visualization on the Internet browser, you may decide to remove some samples or features from the feature table because of outliers or low abundance. The “q2-feature-table” plugin has the “filter-samples” and “filter-features” methods for these purposes. For example, you can remove samples based on their minimum total frequency. The following script removes the samples with a total frequency less than 1000 from feature table created with DADA2 denoising and then it creates a visualization and views it on the Internet browser:

```
qiime feature-table filter-samples \
--i-table dada2/table_yoga_dada2.qza \
--p-min-frequency 1000 \
--o-filtered-table dada2/table_sample_freq_filtered_yoga_dada2.

qiime feature-table summarize \
--i-table dada2/table_sample_freq_filtered_yoga_dada2.qza \
--m-sample-metadata-file data/sample-metadata.tsv \
--o-visualization dada2/table_sample_freq_filtered_yoga_dada2.qzv
qiime tools view dada2/table_sample_freq_filtered_yoga_dada2.qzv
```

Try to study the feature table report after the above filtering.

The “filter-features” method is used to remove low abundance features from a feature table. The following script removes features with a total abundance (across all samples) of less than 20:

```
qiime feature-table filter-features \
--i-table dada2/table_sample_freq_filtered_yoga_dada2.qza \
--p-min-frequency 20 \
--o-filtered-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza
```

```
qiime feature-table summarize \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--m-sample-metadata-file data/sample-metadata.tsv \
--o-visualization dada2/table_feat_sample_freq_filtered_yoga_ \
dada2.qzv
qiime tools view dada2/table_feat_sample_freq_filtered_yoga_dada2. \
qzv
```

You can use “qiime feature-table filter-samples --help” and “qiime feature-table filter-features --help” to learn more about filtering parameters.

The steps of the preprocessing of raw data that we followed above will end up with creation of feature tables and features (OTUs/ASVs) that we will rely on to move to the downstream analysis (taxonomic classification, phylogenetic relationship, alpha and beta diversity analysis, and differential abundance). There are always questions that come up at this point: Which method is better (clustering or denoising)? Which clustering method is the best (de novo, closed-reference, or open-reference clustering)? And, which denoising method is better (DADA2 or deblur)? The right answer from many experts is that: try all of them and adopt the one that works for you. There are number of articles that discussed the pros and cons of each of these methods.

7.3.5 Taxonomic Assignment with QIIME2

As discussed above, one of the main goals of metagenomic studies is to identify the microbial organisms present in a sample using any of the alignment-based classifiers or machine learning classifiers. QIIME2 has alignment-based classifiers, machine learning classifiers, and hybrid classifier methods contained in the “q2-feature-classifier” plugin. The alignment-based classifier methods are “classify-consensus-blast” and “classify-consensus-vsearch”. The machine learning classifier method is “classify-sklearn”, which is used for pre-fitted sklearn-based taxonomy classifiers (any of the classifiers available in scikit-learn python package). You can download a shared pre-fitted classifier; however, it is safer to train yours and use it for taxonomy assignment. Some pre-fitted naïve bayes classifiers and weighted taxonomic classifiers are available at the QIIME2 data resources web page at “<https://docs.qiime2.org/2022.2/data-resources/>” or any newer release. To train your own taxonomy classifier, you can use any of the two training methods provided by “q2-feature-classifier” or “fit-classifier-naive-bayes” to train a naïve bayes classifier or “fit-classifier-sklearn” to train any arbitrary scikit-learn classifier. An alpha hybrid classifier (VSEARCH + sklearn classifier) is provided with “classify-hybrid-vsearch-sklearn” method.

7.3.5.1 Using Alignment-Based Classifiers

The alignment-based classifiers use each of representative sequences generated with clustering or denoising as query sequence to search against the database representative sequences whose taxa are known. The taxonomy assignment is based on the consensus of the BLAST hits at percent identity greater than a predetermined identity threshold. The taxonomy ranks, from the highest to the lowest, are kingdom, phylum, class, order, family, genus, and species. The confidence on the assignment is the fraction of top hits that match

the consensus taxonomy at a given taxonomic level. Consensus is determined at each taxonomic level, beginning from kingdom and stopping when consensus is no longer met above a specified minimum consensus value. Then, the taxonomy is trimmed at this point.

To use an alignment-based classifier, you need to download, extract, and import a representative sequence database and reference taxonomy database as shown above in the de novo clustering section. The following script imports “99_otus.fasta” and “99_otu_taxonomy.txt” reference database from Greensgenes into QIIME2 artifacts, which will be saved in the “input” subdirectory. To keep the files organized, we will create the “taxonomy” subdirectory to store the taxonomy files.

```
mkdir taxonomy
qiime tools import \
--type 'FeatureData[Sequence]' \
--input-path gg_13_8_otus/rep_set/99_otus.fasta \
--output-path taxonomy/99_otus.qza
qiime tools import \
--type 'FeatureData[Taxonomy]' \
--input-format HeaderlessTSVTaxonomyFormat \
--input-path gg_13_8_otus/taxonomy/99_otu_taxonomy.txt \
--output-path taxonomy/99_otu_taxonomy.qza
```

Now, you can run taxonomy classification using the BLAST-based classifier (classify-consensus-blast). The inputs are any representative sequence artifact (generated by any of the clustering methods or denoising methods), the reference sequence artifact, and the reference taxonomy artifact imported in the previous step. The following script uses BLAST-based classifier to assign taxa to the representative sequences generated by DADA2 denoising:

```
qiime feature-classifier classify-consensus-blast \
--i-query dada2/rep-seqs_yoga_dada2.qza \
--i-reference-reads taxonomy/99_otus.qza \
--i-reference-taxonomy taxonomy/99_otu_taxonomy.qza \
--p-perc-identity 0.97 \
--o-classification taxonomy/blast_tax_yoga_dada2.qza \
--verbose
```

If you wish to use the VSEARCH-based method instead, you can use “classify-consensus-vsearch” methods.

```
qiime feature-classifier classify-consensus-vsearch \
--i-query dada2/rep-seqs_yoga_dada2.qza \
--i-reference-reads taxonomy/99_otus.qza \
--i-reference-taxonomy taxonomy/99_otu_taxonomy.qza \
--p-perc-identity 0.97 \
--o-classification taxonomy/vsearch_tax_yoga_dada2.qza \
--verbose
```

7.3.5.2 Using Machine Learning Classifiers

The machine learning taxonomy classifiers use trained model to assign taxa to the representative sequences rather than using the alignment approach. A classifier requires a benchmark training dataset with known taxa for model training. With QIIME2, any of the machine learning methods available in scikit-learn can be used to train a classifier for taxonomy assignment. However, there are also pre-fitted classifiers that can be used instead. In the following, we will use a pre-fitted classifier for the taxonomy assignment, and later, we will train a new model and use it as well.

For pre-fitted classifier, we can use “classify-sklearn” method of “q2-feature-classifier” to assign taxa to the representative sequences that have been obtained from clustering or denoising. First, we need to download a pre-fitted classifier. We can use a classifier pre-trained on GreenGenes database with 99% OTUs using Naive Bayes machine learning method. The pre-fitted classifiers are available at QIIME2 website at “<https://docs.qiime2.org/2022.2/data-resources/>”. Create the subdirectory “classifiers” and download the classifier artifact into it as follows:

```
mkdir classifiers
wget -O "classifiers/gg-nb-99-classifier.qza" \
  "https://data.qiime2.org/2021.11/common/gg-13-8-99-nb-
classifier.qza"
```

Once the download has been completed, use that classifier artifact as an input for “classify-sklearn” method together with the representative sequence artifact generated in the clustering or denoising step. In the following, we will assign taxa to the representative sequences generated by DADA2:

```
qiime feature-classifier classify-sklearn \
--i-classifier classifiers/gg-nb-99-classifier.qza \
--i-reads dada2/rep-seqs_yoga_dada2.qza \
--o-classification taxonomy/nb_tax_yoga_dada2.qza
```

Instead of using a pre-fitted one, we can train a classifier using “feature-classifier” plugin, which has two methods for model fitting: “fit-classifier-naive-bayes” for the training of a naïve bayes classifier and “fit-classifier-sklearn” for the training of any scikit-learn classifier.

Next, we will train a Naive Bayes classifier using GreenGenes reference sequences and then we will use the fitted classifier to assign taxa to the representative sequences generated by a previous clustering or denoising step.

For training any classifier, we need a training dataset with known labels. In the case of taxonomy classification, we need representative sequences with known taxa. For our example, we can use GreenGenes 13_8 97% OTU dataset. Remember that we downloaded GreenGenes database before and stored it in the “gg_13_8_otus” subdirectory. We will use the representative sequences “gg_13_8_otus/rep_set/97_otus.fasta” and their corresponding taxonomic classifications “gg_13_8_otus/taxonomy/97_otu_taxonomy.txt”. Since the

representative sequences are in a FASTA file and taxonomy in text file, we can import both of them as follows:

```
qiime tools import \
--type 'FeatureData[Sequence]' \
--input-path gg_13_8_otus/rep_set/97_otus.fasta \
--output-path inputs/97_otus.qza

qiime tools import \
--type 'FeatureData[Taxonomy]' \
--input-format HeaderlessTSVTaxonomyFormat \
--input-path gg_13_8_otus/taxonomy/97_otu_taxonomy.txt \
--output-path inputs/ref-gg-97-taxonomy.qza
```

After importing the training datasets (sequences and taxonomy) into the “inputs” subdirectory, we can then use the “fit-classifier-naive-bayes” method of the “feature-classifier” plugin to train the naïve bayes classifier and to save it in the “classifiers” subdirectory that we created earlier.

```
qiime feature-classifier fit-classifier-naive-bayes \
--i-reference-reads inputs/97_otus.qza \
--i-reference-taxonomy inputs/ref-gg-97-taxonomy.qza \
--o-classifier classifiers/nb-gg-97-classifier.qza
```

After fitting, you can use the classifier artifact “nb-gg-97-classifier.qza” with the “classify-sklearn” method to assign taxa to our unclassified representative sequences.

```
qiime feature-classifier classify-sklearn \
--i-classifier classifiers/nb-gg-97-classifier.qza \
--i-reads dada2/rep-seqs_yoga_dada2.qza \
--o-classification dada2/nb2_tax_yoga_dada2.qza
```

In the above, we used both alignment-based classifiers (BLAST, VSEARCH) and machine learning classifiers (pre-fitted and fitted) to assign taxa to the unclassified representative sequences. The output of any of these classification steps is an artifact for the classified sequences. Whatever classifier is used for taxonomy assignment, the following is applied to view the taxonomy results. A visualization file can be created from the resulted artifact using “q2-metadata” plugin with “tabulate” method as follows:

Visualizing the BLAST-based taxonomy assignment:

```
qiime metadata tabulate \
--m-input-file dada2/blast_tax_yoga_dada2.qza \
--o-visualization dada2/blast_tax_yoga_dada2.qzv
qiime tools view dada2/blast_tax_yoga_dada2.qzv
```

Visualizing the VSEARCH taxonomy assignment:

```
qiime metadata tabulate \
--m-input-file dada2/vsearch_tax_yoga_dada2.qza \
--o-visualization dada2/vsearch_tax_yoga_dada2.qzv
qiime tools view dada2/vsearch_tax_yoga_dada2.qzv
```

Visualizing the pre-fitted naïve bayes classifier taxonomy assignment:

```
qiime metadata tabulate \
--m-input-file dada2/nb_tax_yoga_dada2.qza \
--o-visualization dada2/nb_tax_yoga_dada2.qzv
qiime tools view dada2/nb_tax_yoga_dada2.qzv
```

Visualizing the trained naïve bayes classifier taxonomy assignment:

```
qiime metadata tabulate \
--m-input-file dada2/nb2_tax_yoga_dada2.qza \
--o-visualization dada2/nb2_tax_yoga_dada2.qzv
qiime tools view dada2/nb2_tax_yoga_dada2.qzv
```

You can compare between the taxonomy classification of the different classifiers. The feature table displayed on the Internet browser has three columns: Feature ID, Taxon, and consensus (for alignment-based classifier) or confidence (for machine learning classifier). The taxon column indicates the taxonomy assignment for each feature (k__ for kingdom, p__ for phylum, c__ for class, o__ for order, f__ for family, g__ for genus, and s__ for species). For example, in Figure 7.15, the naïve bayes classifier predicted the taxa of the first feature up to the family level “f__Coriobacteriaceae;” with a confidence of 0.994, but it did not assign a genus or a species to that feature. However, for the second feature, the classifier predicted taxa up to the species level with confidence of 0.76. The confidence reflects the probability that the taxonomy is correct.

Instead of confidence, alignment-based classifiers provide consensus that is based on the agreement of the alignment hits. For example, a consensus of 1 means that all hits aligned to the feature agreed on the taxa.

A taxonomy bar plot visualization can also be created with “taxa barplot” using the taxonomy predicted by the classifiers and the filtered feature table artifact generated in the clustering or denoising step as input.

Feature ID #q2:types	Taxon categorical	Confidence categorical
0000b3b5727a0706d65375fb42c4bd7b	k_Bacteria; p_Actinobacteria; c_Coriobacteria; o_Coriobacterales; f_Coriobacteriaceae; g__; s__	0.9948011904501861
001a6347c327dfa510dc46cef264f03	k_Bacteria; p_Firmicutes; c_Clostridia; o_Clostridiales; f_Clostridiaceae; g_Clostridium; s_celatum	0.7602659652169912
002264da1158eba91af92e06fd3d6f6e	k_Bacteria; p_Firmicutes; c_Erysipelotrichi; o_Erysipelotrichales; f_Erysipelotrichaceae; g_cc_115; s__	0.9999803257510281
0025f27ed8aa0e026b5ea90a225b190d	k_Bacteria; p_Firmicutes; c_Clostridia; o_Clostridiales; f_Lachnospiraceae; g_Dorea; s__	0.8243829924944165
0031cbc6fb26c4a8a43c1f85bc972ed	k_Bacteria; p_Firmicutes; c_Clostridia; o_Clostridiales; f_Lachnospiraceae	0.938187175229472

FIGURE 7.15 Taxa assigned to features using naïve bayes classifier.

```
qiime taxa barplot \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--i-taxonomy dada2/nb_tax_yoga_dada2.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization dada2/nx_tax_yoga_dada2_barplot.qzv
qiime tools view dada2/nx_tax_yoga_dada2_barplot.qzv
```

The taxonomy barplot (Figure 7.16) provides an interactive graphic interface to view the bacterial taxa distribution in each sample as shown by the color keys. The bar width can be changed by the slider bar on the top left. You can also use the dropdown menus on the top (taxonomic levels, color palette, and Sort Sample by) to change the view and view taxa distribution at specific taxonomic level or sort sample by a taxon or an experimental group.

Earlier, we discussed how to filter the feature table to remove samples or features (with low frequency). However, after viewing the taxonomy report, you may decide to focus on a specific taxon or taxa across all samples. To achieve that you can use “taxa filter-table” to create a new feature table for taxa that you wish to study. Filtering can be applied using that method to retain only specific taxa using “--p-include” or to remove specific taxa using “--p-exclude” or can be applied together. Multiple taxa are provided to the argument as comma-separated list. By default, the terms provided for “--p-include” or “--p-exclude” will match if they are contained in a taxonomic annotation. However, if you need the terms match exactly the complete taxonomic annotation, then you can use “--p-mode exact” parameter as well. Run “qiime taxa filter-table --help” to read more about the usage and parameters. The “taxa” plugin output is a new feature table with the selected taxa. For example, assume that you wish to retain only features that contain a phylum level classification. So, you can use “--p-include p__” as follows:

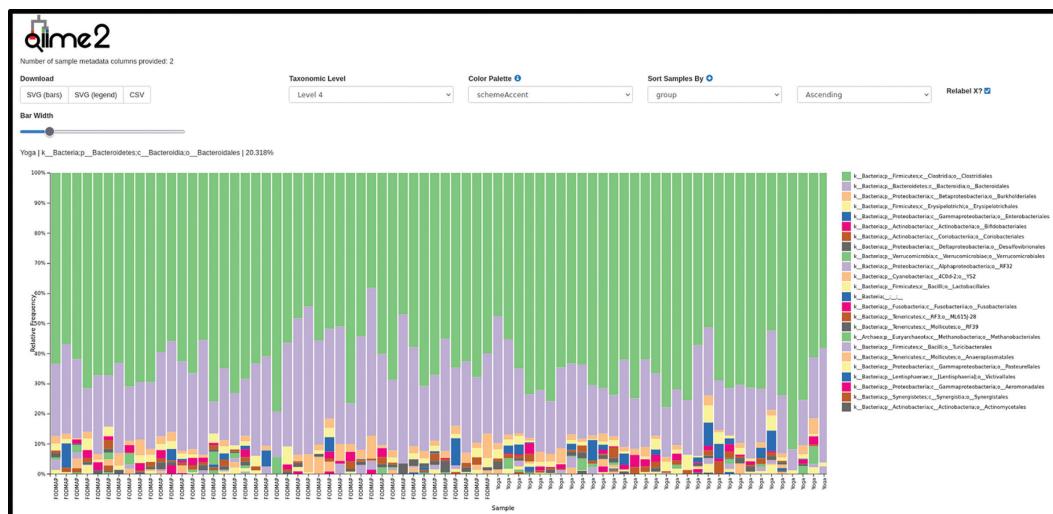


FIGURE 7.16 Taxonomy bar plot (the samples are sorted by group).

```
qiime taxa filter-table \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--i-taxonomy taxonomy/nb_tax_yoga_dada2.qza \
--p-include p__ \
--o-filtered-table dada2/table_phylum_yoga_dada2.qza
qiime taxa barplot \
--i-table dada2/table_phylum_yoga_dada2.qza \
--i-taxonomy taxonomy/nb_tax_yoga_dada2.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization taxonomy/nx_phylum_yoga_dada2_barplot.qzv
qiime tools view taxonomy/nx_phylum_yoga_dada2_barplot.qzv
```

Assume that you intend to explore the species under the “p__Bacteroidetes” phylum. Thus, you can use “p__Bacteroidetes” to retain the features that contain this.

```
qiime taxa filter-table \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--i-taxonomy taxonomy/nb_tax_yoga_dada2.qza \
--p-include "p__Bacteroidetes" \
--o-filtered-table taxonomy/table_Bacteroidetes_yoga_dada2.qza
qiime taxa barplot \
--i-table taxonomy/table_Bacteroidetes_yoga_dada2.qza \
--i-taxonomy taxonomy/nb_tax_yoga_dada2.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization taxonomy/nx_Bacteroidetes_yoga_dada2_barplot.
qzv
qiime tools view taxonomy/nx_Bacteroidetes_yoga_dada2_barplot.qzv
```

A metagenomic study may have a specific design such as groups, treatment, and factorial design. The groups are to be specified in the sample metadata file as discussed above. Grouping samples in an analysis by a group or multiple groups can be achieved by “feature-table group”. The following script creates a feature table in which the samples are grouped by the “group” column of the sample metadata file:

```
qiime feature-table group \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--p-axis sample \
--m-metadata-file data/sample-metadata.tsv \
--m-metadata-column group \
--p-mode sum \
--o-grouped-table taxonomy/groupedby-group-yoga-table.qza
```

You can then create a barplot visualization for the new feature table; however, you need to create a new metadata file in which the group levels (the metadata column which was used for grouping) will be as sample ID. The metadata “taxonomy/group_metadata.tsv” will be as in Figure 7.17.

```
*Untitled - Notepad
File Edit View
SampleID
FODMAP
Yoga
```

FIGURE 7.17 Metadata file for sample grouping (group levels as sample ID).

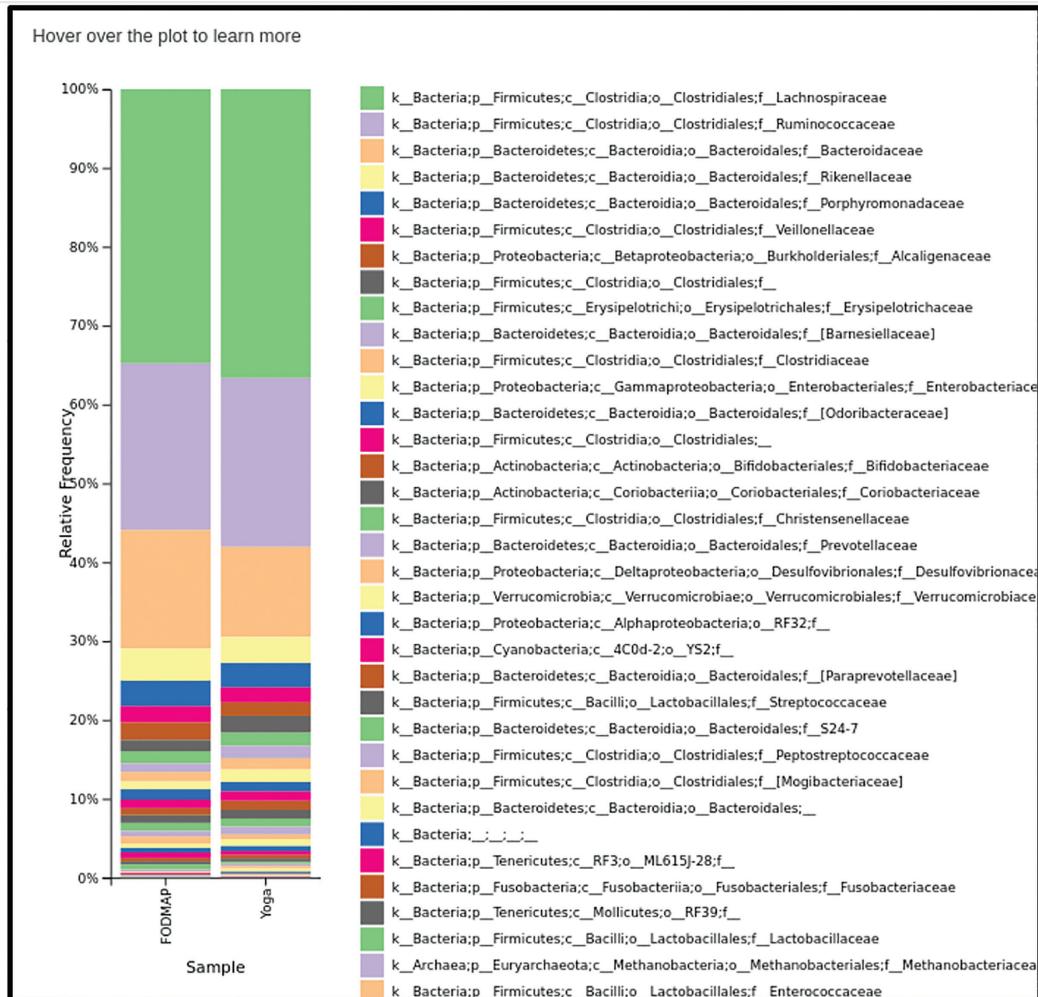


FIGURE 7.18 Taxonomy bar plot for the distribution of taxa in each group.

The barplot can be then created and viewed as follows (Figure 7.18):

```
qiime taxa barplot \
--i-table taxonomy/groupedby-group-yoga-table.qza \
--i-taxonomy taxonomy/nb_tax_yoga_dada2.qza \
```

```
--m-metadata-file taxonomy/group_metadata.tsv \
--o-visualization taxonomy/groupedby-group-yoga-barplot.qzv
qiime tools view taxonomy/groupedby-group-yoga-barplot.qzv
```

7.3.6 Construction of Phylogenetic Tree

The phylogenetic tree is used to visualize the evolutionary relationship between sequences. In the amplicon-based sequence data analysis, a phylogenetic tree is also required for generating some diversity indexes such as Faith's phylogenetic diversity index and UniFrac index.

QIIME2 provides several methods for constructing phylogenetic trees based on the representative sequences generated from clustering or denoising. The methods include alignment-based approaches for the de novo tree construction and a fragment-insertion method that aligns features against a reference tree. It is recommended that the de novo phylogenetic tree is constructed from long sequences only.

7.3.6.1 *De Novo Phylogenetic Tree*

In QIIME2, the de novo phylogenetic tree is constructed by using the following steps:

7.3.6.1.1 Multiple Sequence Alignment

The representative sequences obtained from clustering or denoising step are aligned using “q2-align” plugin with “mafft” method. MAFFT is a program for multiple sequence alignment. For the tree files, we will create the “trees” subdirectory in the project directory.

```
mkdir trees
qiime alignment mafft \
--i-sequences dada2/rep-seqs_yoga_dada2.qza \
--o-alignment trees/aligned-rep-seqs_yoga_dada2.qza
```

7.3.6.1.2 Masking Sites of Low Information

The positions of low phylogenetic information on the sequences are masked to be avoided.

```
qiime alignment mask \
--i-alignment trees/aligned-rep-seqs_yoga_dada2.qza \
--o-masked-alignment trees/masked-aligned-rep-seqs_yoga_dada2.qza
```

7.3.6.1.3 Creating a Tree

An unrooted tree is created from the aligned sequences using Fasttree that infers approximately maximum-likelihood unrooted phylogenetic trees from the sequence alignments.

```
qiime phylogeny fasttree \
--i-alignment trees/masked-aligned-rep-seqs_yoga_dada2.qza \
--o-tree trees/unrooted-denovoTr-tree.qza
```

7.3.6.1.4 Midpoint Rooting

Then, we can define the root of the phylogenetic tree as its midpoint.

```
qiime phylogeny midpoint-root \
--i-tree trees/unrooted-denovoTr-tree.qza \
--o-rooted-tree trees/rooted-denovoTr-tree.qza
```

Now, we have created both unrooted and rooted phylogenetic trees using de novo approach. The tree artifact can be exported into NEWICK tree file format which can be viewed on any tree viewing program such as ETE toolkit. In the following, we will use “export” method of “q2-tools” plugin to export phylogenetic tree data into NEWICK tree files:

```
qiime tools export \
--input-path trees/unrooted-denovoTr-tree.qza \
--output-path tree/unrooted
qiime tools export \
--input-path trees/rooted-denovoTr-tree.qza \
--output-path trees/rooted
```

7.3.6.2 Fragment-Insertion Phylogenetic Tree

A phylogenetic tree based on aligning the representative sequences to reference sequences can be constructed with “q2-phylogeny” using “align-to-tree-mafft-fasttree” method, which takes representative sequences as an input and outputs four artifacts: aligned sequences, masked alignments, unrooted tree, and rooted tree. The following script creates unrooted tree and rooted tree and export them to NEWICK files:

```
mkdir trees
qiime phylogeny align-to-tree-mafft-fasttree \
--i-sequences dada2/rep-seqs_yoga_dada2.qza \
--o-alignment trees2/rep-seqs_yoga_dada2_alignedTr.qza \
--o-masked-alignment trees2/rep-seqs_yoga_dada2_maskedTr.qza \
--o-tree trees2/unrooted-tree-yoga_dada2.qza \
--o-rooted-tree trees2/rooted-tree-yoga_dada2.qza
qiime tools export \
--input-path trees2/unrooted-tree-yoga_dada2.qza \
--output-path trees2/unrooted
qiime tools export \
--input-path trees2/rooted-tree-yoga_dada2.qza \
--output-path trees2/rooted
```

7.3.7 Alpha and Beta Diversity Analysis

Above, we discussed the alpha and beta diversity and the metrics used for each type. In the following, we will show you how to use QIIME2 to compute these metrics.

We will use the “q2-diversity” plugin to generate the phylogenetic and non-phylogenetic diversity metrics. This plugin creates an artifact that contains both alpha and beta diversity

metrics. First, we need to normalize the read counts across sample to adjust for any bias arising from the different sequence depths and to make the comparison meaningful. The normalization is performed by rarefying the count of feature table to a user-specified depth. The lowest read count can be chosen as the user-defined depth. The lowest number of reads is determined from a summary created from the feature table. The lowest count number is then provided to the “-p-sampling-depth” parameter of the “diversity” plugin as a sampling depth for all samples. Once the plugin command is executed, samples are drawn without replacement so that each sample in the resulting table will have a total count equal to that of sampling depth. Then, the alpha and beta metrics are computed. The following script creates summary from the feature table to determine the lowest read number:

```
qiime feature-table summarize \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--o-visualization dada2/table_feat_sample_freq_filtered_yoga_ \
dada2.qzv \
--m-sample-metadata-file data/sample-metadata.tsv
qiime tools view dada2/table_feat_sample_freq_filtered_yoga_dada2.qzv
```

When we study the summary, we can observe that the lowest read number for the samples is 955 sequences. So, we can set the --p-sampling-depth parameter to 955. This step will sub-sample the counts in each sample without replacement so that each sample in the resulting table will have a total count of 955.

The “diversity” plugin requires a phylogenetic tree and feature table artifacts and the sample metadata file as inputs and it outputs the alpha and beta diversity metrics saved into the specified output directory.

```
qiime diversity core-metrics-phylogenetic \
--i-phylogeny trees2/rooted-tree-yoga_dada2.qza \
--i-table dada2/table_feat_sample_freq_filtered_yoga_dada2.qza \
--p-sampling-depth 955 \
--m-metadata-file data/sample-metadata.tsv \
--output-dir diversity-indices
```

The metrics would be saved to the output directory. We can use that metric to explore the microbial composition of sample in the context of the grouping defined in the sample metadata.

We will test for associations between categorical metadata columns and alpha diversity data. We will do that here for the Faith Phylogenetic Diversity (a measure of community richness) and Shannon diversity. The following commands will test for significant differences in the alpha diversity measures of samples:

```
qiime diversity alpha-group-significance \
--i-alpha-diversity diversity-indices/faith_pd_vector.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization diversity-indices/faith-pd-group-significance.qzv
```

```
qiime tools view diversity-indices/faith-pd-group-significance.qzv
qiime diversity alpha-group-significance \
--i-alpha-diversity diversity-indices/shannon_vector.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization diversity-indices/shannon-group-significance.qzv
qiime tools view diversity-indices/shannon-group-significance.qzv
```

These commands will run all-group and pairwise Kruskal-Wallis tests (non-parametric analysis of variance). The visualization files show boxplots and test statistics for each metadata grouping.

We will analyze sample composition (beta diversity group distances) in the context of categorical metadata using PERMANOVA. Note: The qiime diversity beta-group-significance command computes only one metadata grouping at a time, so to test the differences between groups we have to indicate the appropriate column name from the metadata file. In addition, if we call this command with --p-pairwise parameter, it will perform pairwise tests that will allow us to determine which specific pairs of groups are different from one another in terms of dispersion. We will apply a PERMANOVA to test for significant differences of the weighted UniFrac metrics between the samples.

```
qiime diversity beta-group-significance \
--i-distance-matrix diversity-indices/weighted_unifrac_distance_
matrix.qza \
--m-metadata-file data/sample-metadata.tsv \
--m-metadata-column group \
--o-visualization \
    diversity-indices/weighted-unifrac-life-stage-significance.qzv \
--p-pairwise
qiime tools view \
    diversity-indices/weighted-unifrac-life-stage-significance.qzv
```

Finally, we will use the Emperor tool to explore the microbial community composition using principal coordinate analysis (PCoA) plots in the context of sample metadata.

```
qiime emperor plot \
--i-pcoa diversity-indices/weighted_unifrac_pcoa_results.qza \
--m-metadata-file data/sample-metadata.tsv \
--o-visualization diversity-indices/weighted-unifrac-emperor-life-
stage.qzv
qiime tools view diversity-indices/weighted-unifrac-emperor-life-
stage.qzv
```

7.4 SUMMARY

The amplicon-based sequencing is targeting a specific marker gene that is able to distinguish species. Hence, it is used to identify species in a sample that contains multiple microbes such as environmental and clinical samples. The 16S rRNA gene is usually targeted in the

amplicon-based metagenomic analysis because it can identify bacterial species. A region or regions of the gene are amplified using PCR and the amplicon then is sequenced with the high-throughput technologies. The reads are usually for the targeted gene but for several species. The analysis is then focused on identifying the taxonomic groups and their abundance in the sample. After quality control, features unique sequences representing taxonomic groups are obtained either by clustering or denoising. There are three kinds of clustering: de novo clustering, open-reference clustering, and closed-reference clustering. Any of these clustering methods will generate OTUs or operational taxonomic units. On the other hand, denoising attempts to remove base call errors and classification error and it produces ASVs, which are unique features that represent species in the sample. There are three common algorithms for denoising: DADA2, Deblur, and UNOISE2. The most commonly used program for amplicon-based metagenomic data analysis is QIIME2, which implements both clustering methods and denoising methods. To analyze data with QIIME2, raw data must be imported into QIIME2 artifacts. Several analyses can be conducted with QIIME2 including taxonomic group identification and abundance, phylogenetic analysis, and diversity analysis.

REFERENCES

1. Coughlan L, Cotter P, Hill C, Alvarez-Ordóñez A: Biotechnological applications of functional metagenomics in the food and pharmaceutical industries. *Front Microbiol* 2015, 6.
2. Schwartsmann G, Brondani da Rocha A, Berlinck RG, Jimeno J: Marine organisms as a source of new anticancer agents. *Lancet Oncol* 2001, 2(4):221–225.
3. Xiong ZQ, Wang JF, Hao YY, Wang Y: Recent advances in the discovery and development of marine microbial natural products. *Mar Drugs* 2013, 11(3):700–717.
4. Sun Z, Li J, Dai Y, Wang W, Shi R, Wang Z, Ding P, Lu Q, Jiang H, Pei W et al: Indigo Naturalis Alleviates Dextran Sulfate Sodium-Induced Colitis in Rats via Altering Gut Microbiota. *Front Microbiol* 2020, 11: 731.
5. Blaxter M, Mann J, Chapman T, Thomas F, Whitton C, Floyd R, Abebe E: Defining operational taxonomic units using DNA barcode data. *Philos Trans R Soc Lond B Biol Sci* 2005, 360(1462):1935–1943.
6. Westcott SL, Schloss PD: De novo clustering methods outperform reference-based methods for assigning 16S rRNA gene sequences to operational taxonomic units. *PeerJ* 2015, 3:e1487.
7. Rideout JR, He Y, Navas-Molina JA, Walters WA, Ursell LK, Gibbons SM, Chase J, McDonald D, Gonzalez A, Robbins-Pianka A et al: Subsampled open-reference clustering creates consistent, comprehensive OTU definitions and scales to billions of sequences. *PeerJ* 2014, 2:e545.
8. Callahan BJ, McMurdie PJ, Rosen MJ, Han AW, Johnson AJA, Holmes SP: DADA2: High-resolution sample inference from Illumina amplicon data. *Nat Methods* 2016, 13(7):581–583.
9. Nearing JT, Douglas GM, Comeau AM, Langille MGI: Denoising the Denoisers: an independent evaluation of microbiome sequence error-correction approaches. *PeerJ* 2018, 6:e5364.
10. Edgar RC: UNOISE2: improved error-correction for Illumina 16S and ITS amplicon sequencing. *bioRxiv* 2016:081257.
11. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: Basic local alignment search tool. *J Mol Biol* 1990, 215(3):403–410.

12. Rognes T, Flouri T, Nichols B, Quince C, Mahé F: VSEARCH: a versatile open source tool for metagenomics. *PeerJ* 2016, 4:e2584.
13. Cole JR, Wang Q, Fish JA, Chai B, McGarrell DM, Sun Y, Brown CT, Porras-Alfaro A, Kuske CR, Tiedje JM: Ribosomal Database Project: data and tools for high throughput rRNA analysis. *Nucleic Acids Res* 2014, 42(Database issue):D633–642.
14. Olsen GJ: [53] Phylogenetic analysis using ribosomal RNA. In: *Methods in Enzymology*. vol. 164: Academic Press; 1988: 793–812.
15. Bolyen E, Rideout JR, Dillon MR, Bokulich NA, Abnet CC, Al-Ghalith GA, Alexander H, Alm EJ, Arumugam M, Asnicar F et al: Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2. *Nat Biotechnol* 2019, 37(8):852–857.
16. Thompson LR, Sanders JG, McDonald D, Amir A, Ladau J, Locey KJ, Prill RJ, Tripathi A, Gibbons SM, Ackermann G et al: A communal catalogue reveals Earth's multiscale microbial diversity. *Nature* 2017, 551(7681):457–463.

Shotgun Metagenomic Data Analysis

8.1 INTRODUCTION

In the previous chapter, we discussed the amplicon-based metagenomic data analysis which is based on the profiling of a single targeted gene, usually 16S rRNA gene in environmental or clinical samples. Many researchers debate that approach is not metagenomic in nature because it focuses only on a single gene rather than the entire genomes of the microbes in the samples. In this chapter, we will discuss the shotgun sequencing metagenomic approach which involves the sequencing of the entire genomes of the microbes in the samples, and therefore, it provides more insights onto the microbial communities, their genetic profiling, and their impact on hosts and association to the host phenotype. The shotgun sequencing for the metagenomes is rather new but it also emerged as a consequence of the progress in the high-throughput sequencing technologies, which was also followed by the progress in the development of the computational resources and tools that are capable to handle the massiveness and complexity of the metagenomic sequencing data. The shotgun whole-genome metagenomic sequencing and data analysis are now used to quantify the microbial communities and diversity, to assemble novel microbial genomes, to identify new microbial taxa and genes, and to determine the metabolic pathways orchestrated by the microbial community and more.

The metagenomic raw data produced by a high-throughput sequencer is originated from either environmental or clinical samples that contain multiple microbial organisms, including bacteria, fungi, and viruses. Data originated from samples recovered from a host may be contaminated with the host genomic sequences. Multiple samples can also be sequenced in a single run (multiplexing). In the multiplex sequencing, unique barcode sequences identifying each sample are ligated to the DNA fragments in the DNA library preparation step. Some library preparation kits allow multiplexing of hundreds of samples. Illumina has multiple kits for library preparation, including Illumina DNA Prep, (M) tagmentation, which uses bead-linked transposomes in the tagmentation process to randomly

insert transposomes into the metagenomic DNA. The fragmentation is followed by PCR using index primers which enables amplification and subsequent indexing of the sample libraries (barcoding) to allow multiplexing. The library preparation is followed by sequencing and production of the raw data in FASTQ files. The steps of read quality assessment and processing are, to some extent, similar to the steps discussed in Chapter 1. The purpose of the quality control is to reduce sequence biases or artifacts by removing sequencing adaptors, trimming low-quality ends of the reads, and removing duplicate reads. If the DNA is extracted from a clinical sample, an additional quality control step is required which is to remove the contaminating host DNA or non-target DNA sequences. If we need to perform between-sample differential diversity analysis, we may also need to draw a random subsample from the original sample to normalize read counts.

After the step of quality control, there are two strategies that can be followed for the metagenomic raw data. The first one is to assemble the metagenomes using de novo genome assembly method and the second one is an assembly-free approach similar to amplicon-based method. Each of these strategies may address different kinds of questions. The types and algorithms of the de novo assembly were discussed in Chapter 3. However, in the shotgun metagenomics, a new step is introduced. This step is called metagenomic binning, which aims to separate the assembled sequences by species so that the assembled contigs in a metagenomic sample will be assigned to different bins in FASTA files. A bin will correspond to only one genome. A genome built with the process of binning is called Metagenome-Assembled Genome (MAG). Binning algorithms adopt several ways to perform binning. Some algorithms use taxonomic assignment and others use properties of contigs like GC-content of the contigs, nucleotide composition, or the abundance. Binning algorithms use two approaches for assigning contigs to species: supervised machine learning and unsupervised machine learning. Both approaches use similarity scores to assign a contig to a bin. Since many of the microbial species have not been sequenced and hence some of the reads may not map to reference genomes, it is good practice to not rely on mapping to reference genomes. Binning-based nucleotide composition of a contig has been found useful in separating genomes into possible species. The nucleotide composition of a contig is the frequency of k -mers in the contig, where k can be any reasonable integer (e.g., 3, 4, 5, ...). It has been found that different genomes of microbial species have different frequencies that may discriminate the genomes into potential taxonomic groups. A machine learning algorithm like naïve Bayes and other machine learning algorithms are used for taxonomic group assignment. However, features, more powerful than the sequence composition, are required to deal with the complexity in the sequences of contigs. The unsupervised machine learning tools cluster contigs into bins without requiring prior information. There are several binning programs using different algorithms. MetaBAT 2 [1] uses an adaptive binning algorithm that does not require manual parameter tuning as the case with its previous version. Its algorithm consists of multiple aspects, including normalized tetranucleotide frequency (TNF) scores, clustering, and steps to recruit smaller contigs. Moreover, the computational efficiency has been increased compared to the previous version. MaxBin [2] uses nucleotide composition and contig abundance information to group metagenomic contigs into different bins; each bin represents one species. MaxBin algorithm uses tetranucleotide frequencies and scaffold coverage levels to estimate the

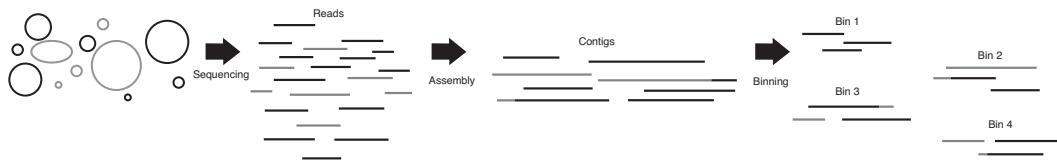


FIGURE 8.1 Reads processing from sequencing to bin formation.

probability that a scaffold belongs to a bin using an expectation-maximization (EM) algorithm. The program also provides statistics, including genome completeness, GC-content, and genome size. Figure 8.1 shows the steps from sequencing to binning.

8.2 SHOTGUN METAGENOMIC ANALYSIS WORKFLOW

The first two steps of the metagenomic data analysis workflow are raw data acquisition and quality control. After the quality control, the raw data can pass through two different steps: (i) de novo assembly and subsequent analysis and (ii) assembly-free analysis. In the following, we discuss these steps with a worked example.

8.2.1 Data Acquisition

The raw shotgun metagenomic data is sequences of the metagenomic DNA extracted from either environmental or clinical samples which usually contain several species of microbes. Depending on the sequencing technology, data can be short reads produced by Illumina and other short-read sequencing technologies or long reads produced either by Pacific Bioscience (PacBio) or by Oxford Nanopore Technology (ONT). The read layout can also be single end or paired end. The raw data is usually provided in FASTQ files. Many researchers uploaded their raw data to a database like NCBI SRA and make it available for public. We will download FASTQ files from the NCBI SRA data for the purpose of demonstrating how analysis is conducted. The run numbers are “ERR1823587”, “ERR1823601”, and “ERR1823608” which contain shotgun metagenomic data of human stool samples from a healthy, a moderate, and a severe sickle cell disease patient, respectively. We will create the directory “shotgun” as the project working directory; then, we will use the SRA-toolkits “fasterq-dump” utility to download the paired-end files in a directory called “fastqdir”.

```
mkdir shotgun; cd shotgun
fasterq-dump --threads 4 --verbose --outdir fastqdir ERR1823587
fasterq-dump --threads 4 --verbose --outdir fastqdir ERR1823601
fasterq-dump --threads 4 --verbose --outdir fastqdir ERR1823608
```

Six FASTQ files will be saved in the “fastqdir”. Use “ls fastqdir/” to display the content of that directory to make sure the files are there.

8.2.2 Quality Assessment and Processing

If you obtained these files directly from the sequencer, it is likely that they may need quality control, which includes both quality assessments using one of the quality assessment programs like FastQC and processing to filter out the low-quality reads, to trim low-quality

ends of the reads, and to remove adaptors and duplicates. Refer to Chapter 1 for detailed information about this step. For multiplexing data, you need to perform demultiplexing before you do the quality control. The multiplexing and demultiplexing are discussed in Chapter 7. The FASTQ files, which we have downloaded, had already been processed and they contain reads of good quality. You can check their quality with FastQC as follows:

```
fqs=$(ls fastqdir/*.fastq)
fastqc $fqs
htmls=$(ls fastqdir/*.html)
firefox $htmls
```

The above commands will display the quality control report of the six FASTQ files on the Firefox browser. Check on the six tabs to study the reports.

8.2.3 Removing Host DNA Reads

Metagenomic data recovered from clinical samples is usually mixed with the host genomic DNA sequences. These sequences, which represent untargeted fraction of data, must be filtered out before the subsequent step of the analysis. Any other untargeted sequences can also be removed following the step of removing host sequences. The process of removing the host sequences begins by aligning raw data to the reference genome of the host. The host sequences will map to the reference genome, whereas the metagenomic reads will not map. Thus, after the mapping process, we can extract the unmapped sequences and store them in separate FASTQ files. For paired-end reads, we will have two FASTQ files representing the raw metagenomic data without the host sequences.

Since the host of our data is human, we will align reads to the human reference genome. We have already discussed read mapping in Chapter 2 and other chapters as well. This time we will use Bowtie2 aligner. We can walk you through the steps without repeating the discussion. The following are the steps to remove the human host sequences from the genomic data.

8.2.3.1 Download Human Reference Genome

You did this step before in Chapter 6. So, if you have the human reference genome and Bowtie2 index saved in your drive, you can use them instead since building the Bowtie2 index may take some time. If you do not have those files stored on your computer, run the following command in your project working directory to download the FASTA sequence of the human reference genome, decompress it, and index it with both Samtools and Bowtie2:

```
mkdir ref; cd ref
wget https://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/hg19.
fa.gz
gunzip -d hg19.fa.gz
samtools faidx hg19.fa
bowtie2-build hg19.fa hg19
cd ..
```

8.2.3.2 Mapping Reads to the Reference Genome

The following Bowtie2 commands will map the files of the samples to the human genome. The SAM files, which contain both mapped and unmapped reads, are saved in “sam” directory.

```
mkdir sam
bowtie2 -p 4 \
-x ref/hg19 \
-1 fastqdir/ERR1823587_1.fastq \
-2 fastqdir/ERR1823587_2.fastq \
-S sam/ERR1823587.sam
bowtie2 -p 4 \
-x ref/hg19 \
-1 fastqdir/ERR1823601_1.fastq \
-2 fastqdir/ERR1823601_2.fastq \
-S sam/ERR1823601.sam
bowtie2 -p 4 \
-x ref/hg19 \
-1 fastqdir/ERR1823608_1.fastq \
-2 fastqdir/ERR1823608_2.fastq \
-S sam/ERR1823608.sam
```

8.2.3.3 Converting SAM to BAM Format

```
samtools view \
-bS sam/ERR1823587.sam \
> sam/ERR1823587.bam
samtools view \
-bS sam/ERR1823601.sam \
> sam/ERR1823601.bam
samtools view \
-bS sam/ERR1823608.sam \
> sam/ERR1823608.bam
```

Remember that the BAM files contain mapped and unmapped reads.

8.2.3.4 Separating Metagenomic Reads in BAM Files

The next step is to remove the unmapped reads in separate files. We will use the “samtools view” and FLAG to separate the unmapped reads.

```
samtools view \
-b -f 12 \
-F 256 \
sam/ERR1823587.bam \
> sam/ERR1823587_unmapped.bam
samtools view \
-b -f 12 \
```

```

-F 256 sam/ERR1823601.bam \
> sam/ERR1823601_unmapped.bam
samtools view \
-b -f 12 \
-F 256 sam/ERR1823608.bam \
> sam/ERR1823608_unmapped.bam

```

The “-f 12” option is used to extract only the unmapped forward and reverse reads and “-F 256” option is used to exclude secondary alignments. Refer to Chapter 2 for FLAG field of the SAM/BAM file.

The above Samtools commands separate unmapped reads, which represent the pure metagenomic data, in the separate BAM files “ERR1823587_unmapped.bam”, “ERR1823601_unmapped.bam”, and “ERR1823608_unmapped.bam”.

8.2.3.5 Creating Paired-End FASTQ Files from BAM Files

Now, we can extract the FASTQ files from the above BAM files; we will extract two FASTQ files from each BAM file. However, before doing that, we need to sort the BAM files by read name using the “samtools sort” command with “-n” option, which sort the paired reads to be next to each other.

```

samtools sort \
-n -m 5G \
-@ 2 sam/ERR1823587_unmapped.bam \
-o sam/ERR1823587_unmapped_sorted.bam
samtools sort \
-n -m 5G \
-@ 2 sam/ERR1823601_unmapped.bam \
-o sam/ERR1823601_unmapped_sorted.bam
samtools sort \
-n -m 5G \
-@ 2 sam/ERR1823608_unmapped.bam \
-o sam/ERR1823608_unmapped_sorted.bam

```

Then, we create FASTQ files from the BAM files and store them in a new directory “fastq_pure” so that we can use them in the next steps of the downstream analysis.

```

Mkdir fastq_pure
samtools fastq -@ 4 sam/ERR1823587_unmapped_sorted.bam \
-1 fastq_pure/ERR1823587_pure_R1.fastq.gz \
-2 fastq_pure/ERR1823587_pure_R2.fastq.gz \
-0 /dev/null -s /dev/null -n
samtools fastq -@ 4 sam/ERR1823601_unmapped_sorted.bam \
-1 fastq_pure/ERR1823601_pure_R1.fastq.gz \
-2 fastq_pure/ERR1823601_pure_R2.fastq.gz \
-0 /dev/null -s /dev/null -n
samtools fastq -@ 4 sam/ERR1823608_unmapped_sorted.bam \

```

```
-1 fastq_pure/ERR1823608_pure_R1.fastq.gz \
-2 fastq_pure/ERR1823608_pure_R2.fastq.gz \
-0 /dev/null -s /dev/null -n
```

The saved FASTQ files contain the reads of the metagenomic data after removing the contaminating human sequences.

If you run FastQC on those files, you will find that the reads length varies between 35 and 151 bp. We can remove any paired reads of length less than 50 bp. Removing such reads from paired-end FASTQ files requires a program or a script that removes reads from both pair of files without leaving singletons that may be rejected by some programs used in the downstream analysis. Thus, we need to filter reads of certain length by using the right program. There may be some programs that can do this but we use a bash script for this purpose. First, change into “fastq_pure” and run the following script to decompress the FASTQ files:

```
cd fastq_pure
for i in $(ls *.gz);
do
  gzip -d ${i}
done
```

Then, open a text editor of your choice such as “vim or nano” and save the following bash script in a file “remove_PE.sh”:

```
vim remove_PE.sh
```

Then, copy the bash script into the file, save it, and exit:

```
#!/bin/sh
#Use: remove_PE.sh R1.fastq R2.fastq 80
#1. Start with inputs
fq_r1=$1
fq_r2=$2
minLength=$3
#2. Find all entries with read length less than minimum length and
print line numbers, for both R1 and R2
awk -v min=$minLength \
'{if(NR%4==2) \
  if(length($0)<min) \
  print NR"\n"NR-1"\n"NR+1"\n"NR+2}' \
$fq_r1 > temp.lines1
awk -v min=$minLength \
'{if(NR%4==2) \
  if(length($0)<min) \
  print NR"\n"NR-1"\n"NR+1"\n"NR+2}' \
$fq_r2@ temp.lines1
```

```

#3. Combine files into one, sort them numerically, and collapse
redundant entries
sort -n temp.lines1 | uniq > temp.lines
rm temp.lines1
outfq1=$(echo $fq_r1| cut -d'.' -f 1)
outfq2=$(echo $fq_r2| cut -d'.' -f 1)
#4. Remove the line numbers recorded in "lines" from both fastqs
awk 'NR==FNR{l[$0];next;}' !(FNR in l)' \
temp.lines $fq_r1 \
> $outfq1-$minLength.fastq
awk 'NR==FNR{l[$0];next;}' !(FNR in l)' \
temp.lines $fq_r2 \
> $outfq2-$minLength.fastq
gzip $outfq1-$minLength.fastq
gzip $outfq2-$minLength.fastq
rm temp.lines

```

Once you have saved the file, you may need to make the file executable by using the Linux command “chmod”:

```
chmod +x remove_PE.sh
```

Then, run the following commands:

```
.
./remove_PE.sh ERR1823587_pure_R1.fastq ERR1823587_pure_R2.fastq 50
./remove_PE.sh ERR1823601_pure_R1.fastq ERR1823601_pure_R2.fastq 50
./remove_PE.sh ERR1823608_pure_R1.fastq ERR1823608_pure_R2.fastq 50
```

Up to this step, we would have removed the host sequences from metagenomic data which are stored in “ERR1823587_pure_R1-50.fastq.gz” and “ERR1823587_pure_R2-50.fastq.gz” for the sample of the healthy person, “ERR1823601_pure_R1-50.fastq.gz” and “ERR1823601_pure_R2-50.fastq.gz” for the moderate sickle cell patient, and “ERR1823608_pure_R1-50.fastq.gz” and “ERR1823608_pure_R2-50.fastq.gz” for severe sickle cell patient. To save some storage space, you can delete the other FASTQ files using “rm *.fastq” and also delete all files in “fastqdir”.

The metagenomic FASTQ files are stored in “fastq_pure” as shown in Figure 8.2. Above, we have deleted the original FASTQ files from “fastqdir” directory and also the intermediate FASTQ files from “fastq_pure” directory. You can also delete the SAM and BAM files from “sam” directory and the reference sequences and indexes from “ref” directory if you want to save storage space. However, you are advised to keep reference genome files in “ref” as you may need to repeat all the steps and indexing usually takes a long time.

8.2.4 Assembly-Free Taxonomic Profiling

We can use the FASTQ files to perform taxonomic profiling without metagenome assembly. This approach employs NGS short or long reads present in the metagenomic samples to assign taxonomic groups by identifying unique genomic regions in the reads. Long reads

allow more accurate taxonomic group assignment. There are several programs for assembly-free classification and profiling of microbial communities in metagenomic samples. Kaiju [3] uses taxonomy and NCBI refseq databases to find maximum matches to the reads on the protein-level using the Burrows–Wheeler transform (BWT). CLARK [4] (CLAssifier based on Reduced K-mers) creates a large index of k-mers of all target sequences and then it removes the common ones among targets so that each target is described by unique k-mers, which are used for taxonomic classification. Kraken [5] creates k-mers from the reads and then it builds taxonomy trees that help discriminate closely related microbes using classification tree and path. Those programs are just examples and there are others with different algorithms. Centrifuge [6] is a rapid classifier that requires a little memory and a relatively smaller index (only 5.8 GB for bacterial, viral, and human genomes) on desktop computers compared to others. Centrifuge uses an indexing system that is based on BWT and the Ferragina–Manzini (FM) index.

Most taxonomy classifiers of the metagenomic data use genomic database of known species to construct an index and then use that index to assign taxa to the metagenomic reads. The majority of the classifiers require a large storage space for database files and a large memory for indexing and classification process. Kaiju and Kraken require a lot of memory (around 128GB–512GB). Therefore, we recommend using these classifiers only if you have enough computational resources. To use any of these classifiers, you need to download and build an index and then to perform the classification.

Kaiju installation instructions are available at “<https://github.com/bioinformatics-centre/kaiju>”. You can install it by running the following command:

```
git clone https://github.com/bioinformatics-centre/kaiju.git
cd kaiju/src
make
```

Then, you need to add its path by adding the following to the “.bashrc” file. You need to replace YOUR_PATH with the program path.

```
export PATH="YOUR_PATH/kaiju/bin":$PATH
```

You must restart the terminal or use “source ~/.bashrc” to make the change active. Run “kaiju” command to check if it has been installed.

Before using kaiju, you need to download the refseq database from the NCBI or you can download it from the kaiju website at “<https://kaiju.binf.ku.dk/server>”. To download it from the NCBI database, use the following:

```
mkdir kaijudb
cd kaijudb
kaiju-makedb -s refseq
```

The download will take a long time and a large storage space. When the database has been downloaded, make sure that “nodes.dmp”, “kaiju_db_refseq.fmi”, and “names.dmp” files are present in the “kaijudb” directory. You may need to decompress

“kaiju_db_refseq_xxxx-xx-xx.tgz”. To classify the short reads in our FASTQ files, you need to run the following:

```
mkdir kaiju_output
kaiju -t kaijudb/nodes.dmp \
-f kaijudb/kaiju_db_refseq.fmi \
-i fastq_pure/ERR1823587_pure_R1-80.fastq.gz \
-j fastq_pure/ERR1823587_pure_R2-80.fastq.gz \
-o kaiju_output/ERR1823587.out \
-a greedy \
-z 4 -v
kaiju -t kaijudb/nodes.dmp \
-f kaijudb/kaiju_db_refseq.fmi \
-i fastq_pure/ERR1823601_pure_R1-80.fastq.gz \
-j fastq_pure/ERR1823601_pure_R2-80.fastq.gz \
-o kaiju_output/ERR1823601.out \
-a greedy \
-z 4 -v
kaiju -t kaijudb/nodes.dmp \
-f kaijudb/kaiju_db_refseq.fmi \
-i fastq_pure/ERR1823608_pure_R1-80.fastq.gz \
-j fastq_pure/ERR1823608_pure_R2-80.fastq.gz \
-o kaiju_output/ERR1823608.out \
-a greedy \
-z 4 -v
```

To learn more about these options, run “kaiju”. The indexing and classification require around 128 GB RAM. We do not recommend using kaiju unless you have enough memory and storage space.

After running the program successfully, you will need to convert the kaiju output file into a summary table using “kaiju2table” command as follows:

```
kaiju2table -t kaijudb/nodes.dmp \
-n kaijudb/names.dmp \
-r taxonomic_level \
-o kaiju_output/ERR1823587_table.tsv \
kaiju_output/ERR1823587.out \
-l taxonomic,levels,separated,by,commas
kaiju2table -t kaijudb/nodes.dmp \
-n kaijudb/names.dmp \
-r taxonomic_level \
-o kaiju_output/ERR1823601_table.tsv \
kaiju_output/ERR1823601.out \
-l taxonomic,levels,separated,by,commas
kaiju2table -t kaijudb/nodes.dmp \
-n kaijudb/names.dmp \
```

```
-r taxonomic_level \
-o kaiju_output/ERR1823608_table.tsv \
kaiju_output/ERR1823608.out \
-l taxonomic,levels,separated,by,commas
```

Run “kaiju2table” to learn about the usage and options of this command.

Most taxonomy classifiers of the metagenomic data follow the same steps: the database downloading and classification. For almost all of them, these steps require large storage space and memory that may not be available on the regular desktop computers. However, if we do not have enough computational resources, we can use Centrifuge which requires relatively small storage space and memory that fits personal computers.

Centrifuge classifier is available at “<https://github.com/infphilo/centrifuge>”. For the updated installation instructions, visit that site. Up to this day, you can install it on Linux using the following commands:

```
git clone https://github.com/infphilo/centrifuge
cd centrifuge
make
sudo make install prefix=/usr/local
```

If it has been installed successfully, no need to do anything else but to use it from any directory. Run “centrifuge -h” to display the usage and options.

As usual, to use Centrifuge classifier, we will begin by building an index. There are several ready-to-use indexes available at <http://www.ccb.jhu.edu/software/centrifuge>. However, Centrifuge also needs sequence and taxonomy files and sequence ID. That can be simplified by using “make” command that can build several standard and custom indices. To do that, find the Centrifuge directory and change into “indices” directory and then run the “make” command as follows:

```
cd indices
make p+h+v      # bacterial, human, and viral genomes [~12G]
make p_compressed # bacterial genomes compressed at the species
level [~4.2G]
make p_compressed+h+v  # combination of the two above [~8G]
```

This command will download the reference taxonomy files and reference genome at assembly levels. The download may take a while depending on the speed of the Internet connection. It is also easier to download a database from Centrifuge homepage, which is available at “<https://ccb.jhu.edu/software/centrifuge/manual.shtml>”. Centrifuge is used to assign taxa to the short reads in the FASTQ files. For the “-x” option, make sure that you provide the database name with the path if it is not in the current path.

```
mkdir centrifuge_out
centrifuge -x p+h+v \
```

```

-1 fastq_pure/ERR1823587_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823587_pure_R2-50.fastq.gz \
--report-file centrifuge_out/ERR1823587-report.txt \
-S centrifuge_out/ERR1823587-results.txt
centrifuge -x p+h+v \
-1 fastq_pure/ERR1823601_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823601_pure_R2-50.fastq.gz \
--report-file centrifuge_out/ERR1823601-report.txt \
-S centrifuge_out/ERR1823601-results.txt
centrifuge -x p+h+v \
-1 fastq_pure/ERR1823608_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823608_pure_R2-50.fastq.gz \
--report-file centrifuge_out/ERR1823608-report.txt \
-S centrifuge_out/ERR1823608-results.txt

```

The results are saved in “*-results.txt” files. Each read classified by Centrifuge results in a single line of output. The output lines consist of eight tab-delimited fields: (1) the read ID (from FASTQ file); (2) sequence ID (from the database sequence); (3) taxonomic ID of the database sequence; (4) classification score (weighted sum of hits); (5) score for the next best classification; (6) two numbers: (i) a number of base pairs of the read that match the database sequence and (ii) the length of a read or the combined length of mate pairs; (7) two numbers: (i) a number of base pairs of the read that match the database sequence and (ii) the length of a read or the combined length of mate pairs; and (8) the number of classifications for this read.

The “*-report.txt” files contain summaries of the identified taxa and their abundances. Each line in the file consists of seven tab-delimited fields: The name of a genome,

name	taxID	taxRank	genomeSize	numReads	numUniqueReads	abundance
Bacteria	2	superkingdom	0	2	2	0.0
Azorhizobium caulinodans	7	species	5369772	40	28	0.0
Buchnera aphidicola	9	species	619958	61	57	0.0
Cellulomonas gilvus	11	species	3526441	27	15	0.0
Dictyogloous	13	genus	0	1	0	0.0
Dictyogloous thermophilum	14	species	1959987	38	35	0.0
Pelobacter carbinolicus	19	species	3665893	48	31	0.0
Shewanella	22	genus	0	1	0	0.0
Shewanella putrefaciens	24	species	4749735	31	10	0.0
Myxococcales	29	order	0	14	0	0.0
Myxococcaceae	31	family	0	7	0	0.0
Myxococcus	32	genus	0	8	0	0.0
Myxococcus fulvus	33	species	10026214	163	100	0.0
Myxococcus xanthus	34	species	9139763	80	41	0.0
Cystobacteraceae	39	family	0	1	0	0.0
Stigmatella	40	genus	0	3	0	0.0
Stigmatella aurantiaca	41	species	10260756	90	53	0.0
Archangium	47	genus	0	2	0	0.0
Archangium gephyra	48	species	12489432	82	52	0.0
Polyangiaceae	49	family	0	4	0	0.0
Chondromyces	50	genus	0	1	0	0.0
Chondromyces crocatus	52	species	11388132	63	57	0.0
Sorangium cellulosum	56	species	13907952	91	57	0.0
Lysobacter	68	genus	0	3	0	0.0
Caulobacter	75	genus	0	3	0	0.0

FIGURE 8.2 Partial centrifuge report for the healthy sample.

taxonomic ID, taxonomic rank (kingdom, genus, family, etc.), genome size in bp, number of reads classified to this genomic sequence including multi-classified reads, number of reads uniquely classified to this genomic sequence, and abundance proportion as shown in Figure 8.2. The Centrifuge report shows that the metagenomic reads have been assigned to taxonomic group in different ranks. This report can be further analyzed to filter the most significant taxa based on their summary statistics.

8.2.4 Assembly of Metagenomes

In the free-assembly microbial profiling, we could assign a taxonomic group to the metagenomic sequences and we could also obtain some useful statistics. In contrast, the assembly-based profiling requires metagenome assembly, which is faced by challenges not present in the assembly of a single genome discussed in Chapter 2. The metagenomic data includes reads for several microbes with different sequence coverage rather than a uniform coverage, which is assumed by typical assemblers to assemble a single genome. Assuming a uniform sequence coverage will allow to distinguish true sequences from errors, identify repeat sequences, and identify allelic variation. That assumption is invalid in metagenome assembly because the coverage of the genome of each species in the sample depends on the abundance of that species in the sample. Since metagenome assembly is performed with de novo approach that uses de Bruijn graph, low sequence coverage may lead to incontiguous path. Assemblers can overcome that challenge by using a short k-mer size but that will also increase the frequencies of identical k-mers in the graph, compromising the assembly quality.

One more challenging problem faced by metagenome assembly is the presence of sub-strains of the same bacterial species and that makes the graph more complex. The metagenome assemblers attempt to overcome these challenges using different strategies. As an example, we will use metaSPAdes [7], which uses de Bruijn graph to form an assembly graph and it also works across a wide range of coverage depths and attempts to maintain the trade-off between the accuracy and continuity of the metagenome assemblies. metaSPAdes is one of the SPAdes programs that we discussed in Chapter 3. Refer to that chapter for SPAdes installation. If you followed the installation instructions of SPAdes in Chapter 3, you would have added its path to the “.bashrc” file. To check if the program is installed and it is on the path, run the following:

```
metaspades.py
```

This will display the usage and options of metaSPAdes program. Otherwise, you may need to install the program following the installation instructions.

The following metaSPAdes command will perform de novo metagenome assembly using the metagenomic FASTQ files as input:

```
mkdir metag_healthy
metaspades.py \
-o metag_healthy \
-1 fastq_pure/ERR1823587_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823587_pure_R2-50.fastq.gz
```

```

-2 fastq_pure/ERR1823587_pure_R2-50.fastq.gz \
--only-assembler \
--threads 4 \
--memory 16 \
--phred-offset 33 \
-k 51
mkdir metag_moderate
metaspades.py \
-o metag_moderate \
-1 fastq_pure/ERR1823601_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823601_pure_R2-50.fastq.gz \
--only-assembler \
--threads 4 \
--memory 16 \
--phred-offset 33 \
-k 51
mkdir metag_severe
metaspades.py \
-o metag_severe \
-1 fastq_pure/ERR1823608_pure_R1-50.fastq.gz \
-2 fastq_pure/ERR1823608_pure_R2-50.fastq.gz \
--only-assembler \
--threads 4 \
--memory 16 \
--phred-offset 33 \
-k 51

```

Run “metaspades.py --help” to read about the usage and options of this program.

Several files are produced in the output directories: “metag_healthy”, “metag_moderate”, and “metag_severe”. The files that contain the assembly sequences are the “contigs.fasta” and the “scaffolds.fasta”. Contigs are made from read overlaps. The contigs are then ordered, oriented, and connected with gaps filled with Ns to form the scaffolds. The K51 directory contains the individual result files for an assembly with 51-mers. However, when multiple K directories are found, the best assembled sequences are the ones that are stored outside these K directories. The directory “misc” contains broken scaffolds.

The file with the “.gfa” extension is in Graphic Fragment Assembly (GFA) file format in which the sequences are represented by lines starting with “S” and the overlaps between sequences are represented by lines starting with “L” as shown in Figure 8.3. The plus (+) and minus (−) signs indicate whether the overlapping sequence is the original or its reverse complement. The value in the form “XM” in a link indicates overlap length.

Thus, the file “assembly_graph_with_scaffolds.gfa” generated by metaSPAdes is the GFA file that represents the final assembly of metagenomes in the sample. SPADes built this assembly graph based on k-mers formed from the reads (vertices) and their overlaps (edges). Then, the assembler resolves paths across the assembly vertices and outputs non-branching paths as contigs.

```

S 6421164 TCATTGTGGTCTGAAACAGATGTGAGACGCCAGTCTGACTGCCATTCTATCATAGATCCGGTAAGGCCATCGGAAGAAAATGGCA>
S 6421172 GTACCAACGGGATGCAACTTATAAATTCCGGACGGCTCAAAGGAAGCATTTCAGCGCTCGGTCAGCTCATTTGACGATCCGAGGGAAATCATT>
S 6421180 GTGACAGGCCCGGCTGATTCCTGATCTGATGCCGTTAAATCTGATCACGGGTTCCGGTGGGGAGATTCAATGAAGGGTTCTGCAGTCCTT>
S 6421188 GATTGGTCTCTGCTGATCACCATATTGTGCTTCAATAAAGTTGAATTGGCTTCAACTTCTGCAATGTTGATGAGCTTACAGTACGAGATGCATCAATTCCATATCTTATCAGTT>
S 6421192 CCACTGACCGGTCACAGCAGATCATGGTGTCTCGAATACATCTTATGGCGATGATGTTCTGCACCTCCAGCTGCAACGCCGGGATTTCCGAAAG>
S 6421194 CTGGTTGGTTATCATGTCATCTTCAAAGGCCCATCTCTGCAATGTTGATAATTGGCTGAGGATTCAGTACGAGATGCATCAATTCCATATCTTATCAGTT>
S 6421204 ATACCTCGGGAGAACAGCGCTCGGATGGAGATTGATGAATAAAAGTAAACATATCTACAAAATTCTGTCAGAACAGTTTTGTAGATGCTAAAT>
S 6421210 64212120 TTAAAATGTTCTCATTTCTCTTCTATCCCGCAGATTGGCGGATTCACGACATCCAACCCATTGGGAGATGCCATCTCCACGGCATCAAGCCAAAGTGGG>
S 6421220 6421244 TTAAAATGTTCTCATTTCTCTTCTATCCCGCAGATTGGCGGATTCACGACATCCAACCCATTGGGAGATGCCATCTCCACGGCATCAAGCCAAAGTGGG>
S 6421244 GTTATGTCACCCATTATAATGTTTCAATTGTTATATGTCAGGGTTTTCGCTCTCCCTGTTTACATTGTTATATTTCCGTATACTTGTTCAT>
S 6421254 GACCGATATCGCTTCTGACTCTGCTGATAGGAAATACCGCTGAGGGCAGTCCCGCTGAAAGCATTACGGCATTTGGCAGTCCGGCATACAGACGC>
S 6421262 TGCCGAAAGACCGCAGATCCAGGAAAGCCTCACAGTACAGATGGGAGCGCTTGGGAGCTGGCGCTCTCCGAAGGGAAATTATGGTTATGCTGGAAG>
S 6421270 GGGCGAGATCCGGAGACGGAAAGCTTACGGCCGGCTGATAGGGCCGGGTGATAGTCCATCTGATCCCGCTGAGTACGAGATAGATCGGC>
S 6421276 6421280 TGCGATACCCGATGGGAAAGATCAGCTATGAATACGACTTAAATGATAATTGTTCTACATACGGCAGACACCAGATCCGTCCTGGTATGAAAGCCAT>
S 6421288 GGTGTCCTAAACAGTGGTAAATTGCAATTGTTCTACATACGGCAGACACCAGATCCGTCCTGGTATGAAAGCCATGATATTTCATAAAG>
L 133089 - 24069 - 51M
L 133089 - 5382192 + 51M
L 6207015 - 5230392 + 51M
L 6207015 + 5230392 + 51M
L 1009 - 1701458 - 51M
L 1009 - 6401035 - 51M
L 5368574 - 20363 + 51M
L 5368574 - 4502917 + 51M
L 6371006 - 6381318 - 51M
L 6371006 - 6382696 - 51M
L 3869 + 25163 - 51M
L 6250593 + 25163 - 51M
L 94173 - 5450433 - 51M
L 94173 - 6353154 - 51M

```

FIGURE 8.3 Graphic Fragment Assembly (GFA) file format.

This GFA file can be visualized by graph visualization programs like Bandage [8]. The Bandage program is available at “<http://rrwick.github.io/Bandage/>” and can be downloaded and installed on Linux, Windows, and Mac OX. Visualizing a graph file will give you an idea about the assembly quality. You can zoom in and out and do many operations on the graph file. Moreover, there are good graph visualization packages in Python and R such as igraph [9], which is available in both programming platforms. To read more about Bandage and igraph, refer to their user manuals which are available on their web pages.

8.2.5 Assembly Evaluation

MetaSPAdes produced several assemblies for different species in a single scaffolds file. Those assemblies can be evaluated by using an evaluation program like “metaquast.py” [10], which is a Python-based program that can generate report showing the quality of the assemblies. For the installation instructions of this program, visit “<http://cab.cc.spbu.ru/quast/manual.html>”. To generate report for the scaffolds for each of the three samples, you can run the following:

```

metaquast.py -t 4 \
-m 500 \
metag_healthy/scaffolds.fasta \
metag_moderate/scaffolds.fasta \
metag_severe/scaffolds.fasta \
-o output

```

This will generate an HTML report in the “output” directory. The other directories and files are linked to this report when it is displayed on the Internet browser. You can display “report.html” file by using “firefox report.html”.

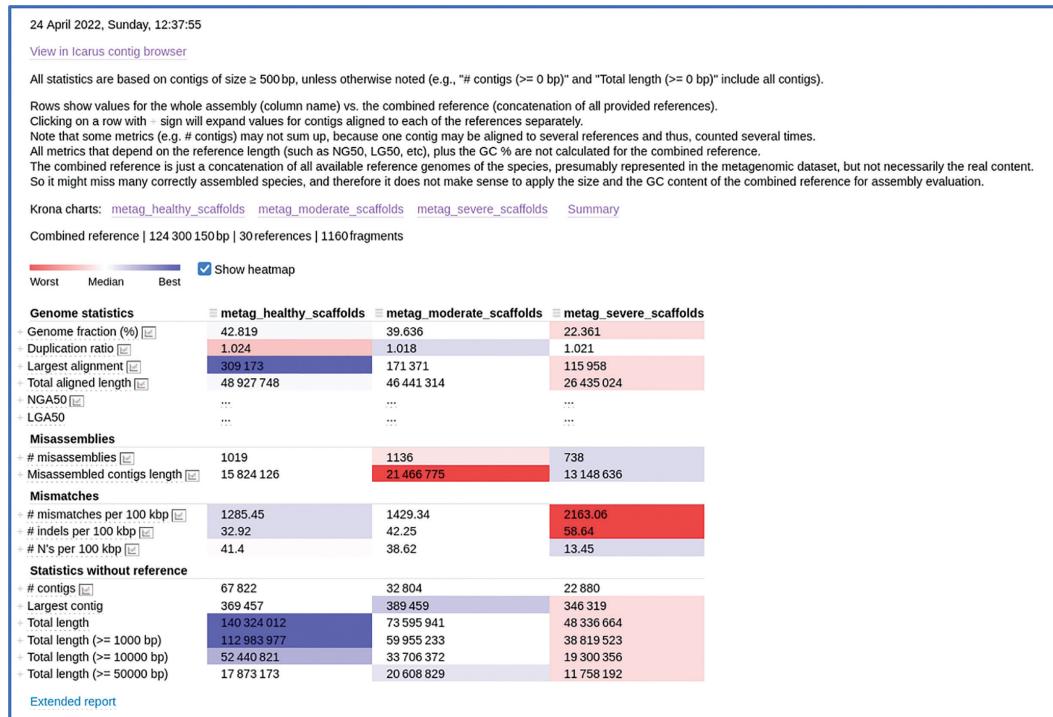


FIGURE 8.4 An assemblies' evaluation report generated with metaquast.py.

The assemblies' evaluation report contains important statistics that reflect the quality of the assemblies in the file. Figure 8.4 shows partial evaluation report of the three samples. The colored heatmap indicates the quality from the worst (red) to the best (blue). On the top, there are links that can take to each sample graph as shown in Figure 8.5. The graphs show the key identified bacterial taxa and their abundance. Refer to the program users' manual, which is available at "<http://cab.cc.spbu.ru/quast/manual.html>", to read more about the program use and the different report sections, refer to Chapter 3 to read more about the de novo assembly evaluation metrics.

8.2.6 Mapping Reads to the Assemblies

We have already created the metagenomic FASTQ files, which are produced from the reads unmapped to the host reference genome. Then, we used a de novo assembler to produce an assembly (scaffolds.fasta) for each sample that may contain genomic sequences of several microbes. In this step, we will use an aligner to map the reads in the FASTQ files to the respective assembly. For this purpose, we can use Bowtie2 aligner. First, we need to build an index for the "scaffolds.fasta" for each sample and then we will use it to align the reads in FASTQ files. Now, let us create a directory named "assemblies" in the main project directory and copy scaffolds FASTA files from the three sample directories into this new directory with new file names as follows:

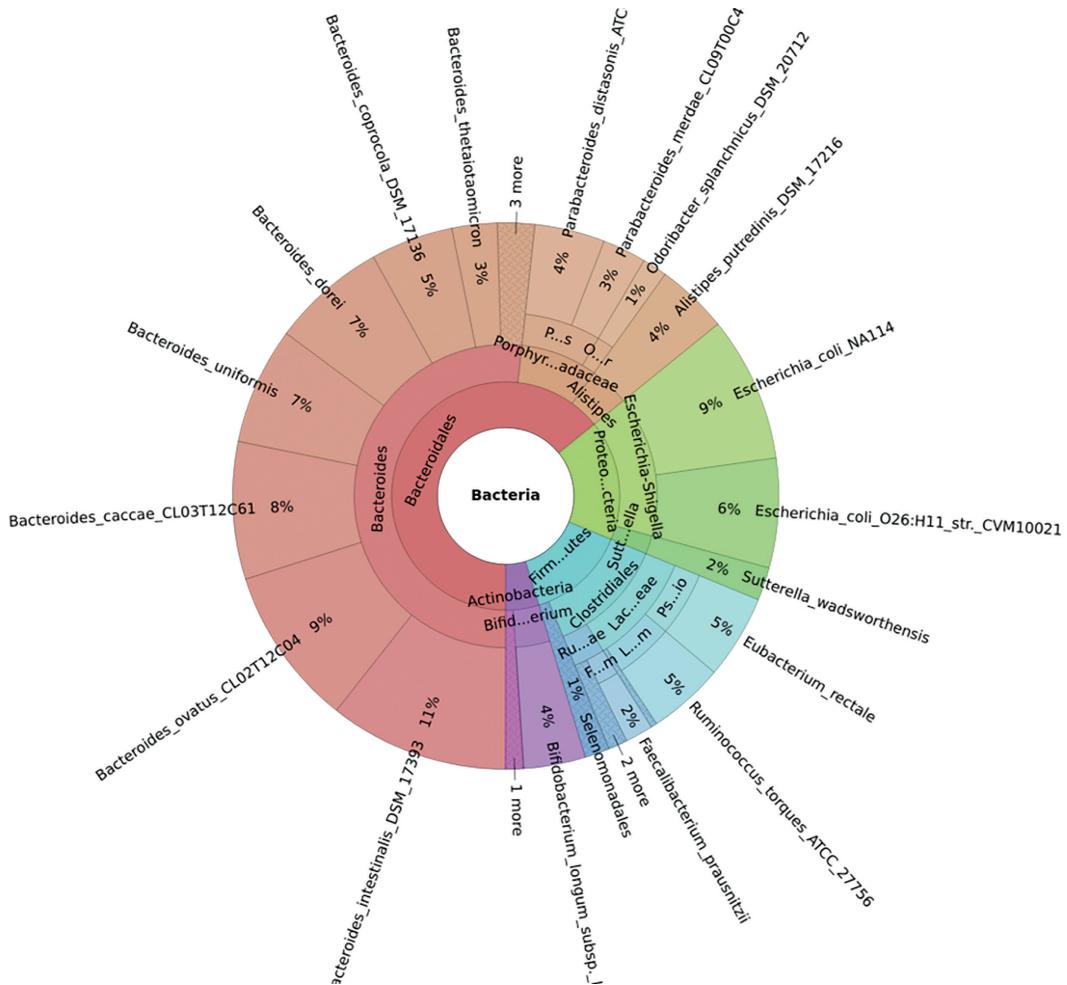


FIGURE 8.5 The metagenomic composition of the healthy individual.

```
mkdir assemblies
cp metag_healthy/scaffolds.fasta assemblies/healthy_scaffolds.fasta
cp metag_moderate/scaffolds.fasta assemblies/moderate_scaffolds.fasta
cp metag_severe/scaffolds.fasta assemblies/severe_scaffolds.fasta
```

Then, we can use Samtools and Bowtie2 to build an index for each “scaffolds.fasta” file of each sample.

```
cd assemblies  
cd assemblies  
for i in $(ls *.fasta);
```

```

do
    samtools faidx ${i}
done
bowtie2-build healthy_scaffolds.fasta healthy
bowtie2-build moderate_scaffolds.fasta moderate
bowtie2-build severe_scaffolds.fasta severe

```

Once the index has been built, we can use Bowtie2 to align the FASTQ reads to their respective “scaffolds.fasta”.

```

mkdir sam_assemblies
bowtie2 --sensitive-local \
    -p 4 \
    -x assemblies/healthy \
    -1 fastq_pure/ERR1823587_pure_R1-50.fastq.gz \
    -2 fastq_pure/ERR1823587_pure_R2-50.fastq.gz \
    -S sam_assemblies/ERR1823587_healthy.sam
bowtie2 --sensitive-local \
    -p 4 \
    -x assemblies/moderate \
    -1 fastq_pure/ERR1823601_pure_R1-50.fastq.gz \
    -2 fastq_pure/ERR1823601_pure_R2-50.fastq.gz \
    -S sam_assemblies/ERR1823601_moderate.sam
bowtie2 --sensitive-local \
    -p 4 \
    -x assemblies/severe \
    -1 fastq_pure/ERR1823608_pure_R1-50.fastq.gz \
    -2 fastq_pure/ERR1823608_pure_R2-50.fastq.gz \
    -S sam_assemblies/ERR1823608_severe.sam

```

We can notice that there are some statistics when the alignment process finishes for each sample.

We will convert SAM files to bam files and then we will sort the alignments in the BAM files.

```

cd sam_assemblies
samtools view -S -b ERR1823587_healthy.sam > ERR1823587_healthy.bam
samtools view -S -b ERR1823601_moderate.sam > ERR1823601_moderate.bam
samtools view -S -b ERR1823608_severe.sam > ERR1823608_severe.bam

for i in $(ls *.bam);
do
    samtools sort -@ 4 ${i} -o ${i}.sorted
done

```

We need to index the sorted BAM files using “samtools index” command.

```
for i in $(ls *.sorted);
do
    samtools index -@ 4 ${i}
done
```

Then, we will use “samtools idxstats” to generate some statistics from the sorted BAM files.

```
samtools idxstats ERR1823587_healthy.bam.sorted > ERR1823587_healthy_stat.txt
samtools idxstats ERR1823601_moderate.bam.sorted > ERR1823601_moderate_stat.txt
samtools idxstats ERR1823608_severe.bam.sorted > ERR1823608_severe_stat.txt
```

The output of the “samtools idxstats” command is a TAB-delimited file with each line consisting of the reference sequence name, sequence length, number of mapped read-segments, and number of unmapped read-segments. From those files, we can generate abundance table similar to the OTU (operation taxonomic units) generated from clustering of the amplicon-based reads in Chapter 7. For this purpose, we can use “get_count_table.py” script, which can be cloned from GitHub using the following command:

```
git clone https://github.com/metajinomics/mapping_tools.git
```

Then, we can use that Python 2 script to generate an abundance table for each sample. So, if you do not have Python 2 installed on your computer, you may need to install it.

```
python2 mapping_tools/get_count_table.py ERR1823587_healthy_stat.txt > ERR1823587_healthy_count.txt
python2 mapping_tools/get_count_table.py ERR1823601_moderate_stat.txt > ERR1823601_moderate_count.txt
python2 mapping_tools/get_count_table.py ERR1823608_severe_stat.txt > ERR1823608_severe_count.txt
cd ..
```

We will use the output of this script for binning in the next step.

8.2.7 Binning

Above, we discussed binning as the process of separating the sequences into bins that represent the most likely taxa. There are many programs that can do this job including metabat2, CONCOCT, and MaxBin. Here, we will use metabat2 as an example. Metabat2 is easier to install on Anaconda or Miniconda.

```
conda install -c biconda metabat2
conda install -c bioconda/label/cf201901 metabat2
```

Before we perform the taxonomic binning, we need to generate sequence depth from the sorted BAM files produced by mapping the metagenomic FASTQ reads to the de novo assemblies. For this purpose, we will also need the tables produced by “get_count_table.py” script above as an input with the sorted BAM file for the “jgi_summarize_bam_contig_depths” function to produce a file of five columns: contig name, contig length, total average depth, mean depth, and variance.

```
mkdir stats_metabat
jgi_summarize_bam_contig_depths \
    --outputDepth stats_metabat/healthy_depth.txt \
    sam_assemblies/ERR1823587_healthy.bam.sorted
jgi_summarize_bam_contig_depths \
    --outputDepth stats_metabat/moderate_depth.txt \
    sam_assemblies/ERR1823601_moderate.bam.sorted
jgi_summarize_bam_contig_depths \
    --outputDepth stats_metabat/severe_depth.txt \
    sam_assemblies/ERR1823608_severe.bam.sorted
```

Then, we can perform binning on the contigs.fasta produced by de novo assembly above. We can copy these files in a new directory “binning” with new names.

```
mkdir binning
cp metag_healthy/contigs.fasta binning/healthy_contigs.fasta
cp metag_moderate/contigs.fasta binning/moderate_contigs.fasta
cp metag_severe/contigs.fasta binning/severe_contigs.fasta
```

The next step is to separate the contigs in the contigs files into bins; each bin represents a species. The bins of the three samples are saved in different subdirectories inside “binning” directory.

```
mkdir binning/healthy
metabat2 -i binning/healthy_contigs.fasta \
    -a stats_metabat/healthy_depth.txt \
    -o binning/healthy/healthy \
    -t 4 -v --seed 123
mkdir binning/moderate
metabat2 -i binning/moderate_contigs.fasta \
    -a stats_metabat/moderate_depth.txt \
    -o binning/moderate/moderate \
    -v --seed 123
mkdir binning/severe
metabat2 -i binning/severe_contigs.fasta \
    -a stats_metabat/severe_depth.txt \
    -o binning/severe/severe \
    -v --seed 123
```

The “-i” option specifies the input contigs FASTA file, “-a” option specifies the depth file that contains contig depth averages and variances, “-o” specifies the output path and prefix, “-v” for verbose, and “--seed” specifies a seed integer to replicate the same results.

Up to this point, we have performed the taxonomic binning successfully and now we have separated genomes for each potential species in the metagenomic sample. However, we do not know the qualities of these genomes and to which microbial species they belong. So, the next step, we must evaluate these genomic sequences and assess their completeness with regard to protein-coding genes and their annotations.

8.2.8 Bin Evaluation

The binning quality is usually assessed with CheckM [11], which includes a collection of tools for assessing the quality of the genome sequence separated from metagenomes and also to assess the quality of genomes recovered from single cells and isolates. CheckM provides estimates of genome completeness and contamination in addition to plots and other important reports. For this software installation, visit “<https://github.com/Ecogenomics/CheckM/wiki>”. On Linux, it requires HMMER, Prodigal, and Pplacer programs to be installed and added to the system path.

```
sudo apt update
sudo apt install hmmer
sudo apt install prodigal
```

You need to follow the installation instructions at Pplacer home page, which is available at “<https://matsen.fhcrc.org/pplacer/>”, and add it to the Linux path. Then, you can install CheckM with the following commands:

```
pip3 install numpy
pip3 install matplotlib
pip3 install pysam
pip3 install checkm-genome
```

You can also install CheckM on Anaconda using:

```
conda install -c bioconda checkm-genome
conda install -c bioconda/label/cf201901 checkm-genome
```

Now, we can run CheckM commands to assess the completeness and contamination of the genome bins by using lineage-specific marker sets. This workflow consists of several steps that include placing bins in the reference genome tree, assessing phylogenetic markers found in each bin, and inferring lineage-specific marker sets for each bin. These steps are done with multiple CheckM commands but they can also be done in a single step by using “lineage_wf” command.

```

mkdir checkM_out
mkdir checkM_out/healthy
checkm lineage_wf \
-t 4 \
-x fa \
-f checkM_out/healthy_checkm_report.txt \
binning/healthy \
checkM_out/healthy
mkdir checkM_out/moderate
checkm lineage_wf \
-t 4 \
-x fa \
-f checkM_out/moderate_checkm_report.txt \
binning/moderate \
checkM_out/moderate
mkdir checkM_out/severe
checkm lineage_wf \
-t 4 \
-x fa \
-f checkM_out/severe_checkm_report.txt \
binning/severe \
checkM_out/severe

```

The report (Figure 8.6) shows the bin ID, marker lineage (taxonomic rank), # genome (number of genomes used to infer marker sets), # marker (number of marker genes), # marker set (number of sets within the inferred markers), 0–5+ (number of times each marker gene is identified), completeness (presence/absence of marker genes), and strain heterogeneity (high heterogeneity indicates the contamination is from one or more closely related organisms).

In Figure 8.6, for the moderate sample, we can notice that for the bin “moderate.4”, there are 5449 bacterial genomes that were used to infer 104 markers genes; only 6 genes were inferred in the bin, the completeness is 10.34, and there was no contamination. For more details about the use of CheckM and report, refer to the program home page at “<https://ecogenomics.github.io/CheckM/>”.

8.2.9 Prediction of Protein-Coding Region

This step is to annotate the single genomes recovered by binning or metagenomic assemblies with potential gene locations by predicting the open reading frames (ORFs). The gene

Bin Id	Marker lineage	# genomes	# markers	# marker sets	0	1	2	3	4	5+	Completeness	Contamination	Strain heterogeneity
moderate.4	k_Bacteria (UID203)	5449	104	58	98	6	0	0	0	0	10.34	0.00	0.00
moderate.5	k_Bacteria (UID203)	5449	104	58	99	5	0	0	0	0	8.62	0.00	0.00
moderate.1	k_Bacteria (UID203)	5449	104	58	100	4	0	0	0	0	6.90	0.00	0.00
moderate.2	k_Bacteria (UID203)	5449	103	58	100	3	0	0	0	0	5.17	0.00	0.00
moderate.3	root (UID1)	5656	56	24	55	1	0	0	0	0	4.17	0.00	0.00

FIGURE 8.6 Genome completeness evaluation report generated by CheckM.

annotations and polypeptides and ORFs are written to files. Gene annotation of a new genome assembly is an important step. Since bacteria have no introns, prediction of ORFs is easier than in the eukaryotic genome. There are many programs for ORF prediction, but Prodigal [12] is the most commonly used one. We have installed Prodigal above. Prodigal can predict ORFs in any genomic sequences. Thus, we can predict the ORFs in assemblies separated by binning. In the following, we will predict the ORFs in one of the assemblies recovered from the sample of the patient with severe sickle cell disease.

```
prodigal -a prod_out/healthy.faa \
-d prod_out/healthy.fnt \
-o prod_out/healthy.gbk \
-s prod_out/genes.gff \
-i binning/severe/severe.1.fa \
-p single
```

The “-a” option specifies the FASTA file name for the polypeptides or proteins translated from the predicted ORFs. The “-d” option specifies the FASTA file name of the nucleotide sequences that represent the predicted ORFs. The “-o” option specifies the predicted ORF as features in GenBank format. The “-s” option specifies the gene annotation in GFF (general feature format). The “-i” option specifies the input file which is the assembly. The “-p” option specifies the procedure, which is either “single” for a single assembly or “meta” for metagenomic assembly that may include genomes of multiple species.

8.3 SUMMARY

The metagenomic DNA is isolated from environmental samples or clinical samples in which several microbes are present. Unlike targeted gene sequencing, shotgun metagenomic sequencing allows researchers to sequence the whole genomes of all organisms present in a sample and to evaluate the microbial diversity and abundance.

Shotgun metagenomic sequencing attempts to sequence the whole genomes of a large diverse number of microbes, each with a different genome size. Long reads produced by PacBio and Oxford Nanopore are preferred. However, they usually have higher error rate than the short reads. Since there are several species in the metagenomic sample, there must be a sufficient sequencing depth to allow assembling the genomes of all species in the sample.

Before analysis, we should make sure that we have fixed any quality problem by trimming adaptors, filtering out low-quality reads, and removing technical sequences. In the case of clinical samples, we should also remove the host DNA by aligning reads to the host genome and then separate the unaligned reads in new FASTQ files to be used in the analysis. There are two approaches for the shotgun metagenomic data analysis: the assembly-free and de novo assembly. The assembly-free approach does not require assembling the genomes of the species in the sample; it uses reads present in the metagenomic samples to assign taxonomic groups by identifying unique genomic regions in the reads. Most of the programs used for taxonomy assignment require a large amount of memory and storage space. The second approach uses de novo algorithms to assemble the genomes of the

species in the sample. This method provides information on the total genomic DNA from all species in a sample. After running de novo assembly, binning is used to cluster contigs that apparently originated from the same species population. The recovered genome sequences can be annotated by identifying protein-coding sequence of the ORFs.

The shotgun metagenomic sequencing is widely used to study unculturable microorganisms that are difficult to culture and analyze.

REFERENCES

1. Kang DD, Li F, Kirton E, Thomas A, Egan R, An H, Wang Z: MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ* 2019, 7:e7359.
2. Wu Y-W, Tang Y-H, Tringe SG, Simmons BA, Singer SW: MaxBin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome* 2014, 2(1):26.
3. Menzel P, Ng KL, Krogh A: Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nat Commun* 2016, 7(1):11257.
4. Ounit R, Wanamaker S, Close TJ, Lonardi S: CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics* 2015, 16(1):236.
5. Wood DE, Salzberg SL: Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol* 2014, 15(3):R46.
6. Kim D, Song L, Breitwieser FP, Salzberg SL: Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Res* 2016, 26(12):1721–1729.
7. Nurk S, Meleshko D, Korobeynikov A, Pevzner PA: metaSPAdes: a new versatile metagenomic assembler. *Genome Res* 2017, 27(5):824–834.
8. Wick RR, Schultz MB, Zobel J, Holt KE: Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* 2015, 31(20):3350–3352.
9. Csárdi G, Nepusz T: The igraph software package for complex network research. In: 2006.
10. Mikheenko A, Prjibelski A, Saveliev V, Antipov D, Gurevich A: Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics* 2018, 34(13):i142–i150.
11. Parks DH, Imelfort M, Skennerton CT, Hugenholtz P, Tyson GW: CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome Res* 2015, 25(7):1043–1055.
12. Hyatt D, Chen G-L, LoCascio PF, Land ML, Larimer FW, Hauser LJ: Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinfor* 2010, 11(1):119.

Index

- 16S rRNA gene 253, 260, 263
- ABySS 92–96
- acetylation 213
- adaptor 4–9, 13, 18
- adaptor dimer 18, 31–33, 46, 218
- adenine 1–2, 13
- adenosine triphosphate 5
- alpha diversity 261–262, 299
- alternative splicing 164–165, 172
- ammonium persulfate 5
- amplicon 253–257
- ANNOVAR 145, 151–159, 161
- archaea 49
- ASCII 12, 14, 23, 67–68
- assembly-free 304–305, 310–311, 325
- Augustus 104
- bacteria 49, 97, 105, 160, 253–254, 303, 325
- bacteriophage 3, 46
- base caller 11
- base calling 6, 9, 11, 23, 85, 135, 256
- Base Quality Score Recalibration *see* BQSR
- base substitution 109–110, 114, 160
- Basic Local Alignment Search Tool *see* BLAST
- BCFTools 84–85
- beta diversity 261–263, 289, 298–300
- binning 304–305, 321–326
- BioProject 20, 271–272, 274
- BioPython 104
- BLAST 55, 104, 146, 258–259, 287
- Bowtie 58, 70–71, 75, 87, 215
- BQSR 130, 135–139
- Bray-Curtis distance 261
- Burrows-Wheeler transform *see* BWT
- BUSCO 100, 103–108
- BWA 58, 70–75
- BWA-backtrack 72, 74
- BWA-MEM 72–74, 168
- BWA-SW 72, 74
- BWT 56, 58–63, 65, 71–72, 75, 86
- CD-HIT 255
- centrifuge 311, 313–315, 326
- CheckM 323–324, 326
- chimeric 77, 81, 86, 172, 284
- chimeric read 81, 172, 284
- ChIP-Seq 214–217
- ChIPseeker 230, 236, 251
- chromatin 213–215
- chromatin immunoprecipitation *see* ChIP-Seq
- chromatin remodeling 213
- chromatography 3
- CIGAR string 67–68, 82, 86
- circular consensus sequences 9
- CLARK 311, 326
- closed-reference 255, 278, 280–282, 289, 301
- clustering 193–194, 254–255, 257–258
- CNV 109–110
- consensus sequence 83, 85, 89–92, 114–115, 160
- contig 83, 89–91, 93–100
- copy number variation *see* CNV
- coverage 89–91
- Crick, Francis 3
- Crohn’s disease 144
- cytogenetic bands 151, 154
- cytosine 1–2, 13, 213, 247
- de Bruijn 90–93, 97, 107–108, 126, 137, 315
- de novo 89–95, 97–101, 103–105, 297–298
- de novo clustering 255, 280, 282, 285, 290, 301
- deep sequencing 89–90, 107
- defline 17, 51, 54
- deletion 68–70, 82, 86, 109–112
- dereplication 279–281
- DESeq 174, 204, 210, 212
- DGEList 182–184
- differential expression 165–166, 176–177, 180, 185
- DNA ligase 6
- DOT language 95
- double-stranded DNA *see* dsDNA
- dsDNA 9
- dynamic programming 55

- EBI 50, 265
 EdgeR 174–175
 electrophoresis 3–4
 entropy 126
 epigenetics 4–5, 9, 11, 45, 49, 64, 86
 eQTL 165–166
escherichia coli 20, 49, 94
 ETE toolkit 298
 Eulerian path 90, 92, 107
 evenness 261–262
 exon 2, 52, 63, 77, 144
 Faith's phylogenetic diversity 261–262, 297
 Fastp 44, 46–47
 Fastqc 18–23, 32, 36–44
 Fasttree 297–298
 feature table 258, 261, 278, 280–283
 FeatureCounts 173, 212
 Ferragina-Manzini 311
 flow cell 4, 7–10, 14, 18, 25–27, 66, 134
 FM-index 56, 62, 71, 75, 86–87
 FPKM 174–175, 206–207, 210
 frameshift 110, 143–144, 148, 150, 159
 FreeBayes 114, 127, 129, 143
 Full-text Minute-space *see* FM-index
 fungi 49, 97, 303
 fusion 165–166, 211
 GATK 66, 114, 127, 129–132,
 134–143, 160
 GC content 20, 23–24, 28–29, 46–47
 GenBank 50–52, 325
 gene expression 163–167
 gene ontology *see* GO
 gene transcription 163–164, 218, 241
 gene transfer format *see* GTF
 General Feature Format *see* GFF
 genetic drift 114
 genetic mosaicism 113
 Genome Reference Consortium *see* GRC
 Genome Variant Call Format *see* gVCF
 GFF 51–52
 ggplot2 104
 Gilbert, Walter 3
 global sequence alignment 55
 GO 66, 165, 202–203, 211–212, 239–241, 250
 graphviz 95
 GRC 50
 greedy 90–91, 107, 255, 312
 greengenes database 255, 281, 291
 GTF 51–53
 guanine 1–2, 13, 213, 247
 gVCF 130, 137–138
 Hamiltonian path 90–92, 107
 Hamming distance 65, 255, 257, 260
 haplotype 83, 104, 112, 114, 125–127
 HaplotypeCaller 114, 130, 137
 hard filtering 141–142
 hashing xiv 65, 93
 hash table 65, 86, 148
 heatmap 25, 193–194, 200–202, 233, 318
 helicase 3
 heuristic clustering 255
 hierarchical clustering 193, 255
 high-throughput sequencing *see* HTS
 histone 213, 216–217, 235–236, 243
 HMMER 104, 323
 homology 144, 251
 HTS 13, 34, 45, 69–70
 HTSeq-count 173–174, 210
 hydroxyl 5, 7
 Icarus 102, 104
 iGenomes 169
 IGV 142–143
 InDel 114, 116, 121
 insertion 63, 68–70
 insertion-deletion *see* InDel
 Integrated Genomics Viewer *see* IGV
 Ion Torrent 6–7, 97–98
 IQR 24
 Jaccard distance 261–262
 Jukes-Cantor 260
 k-mers 33, 65, 91–93, 97, 107
 Kaiju 311–312, 326
 KEGG 165, 202–203, 211–212, 239, 241–242, 250
 Kraken 311, 326
 Kruskal-Wallis 300
 Kyoto Encyclopedia of Genes and Genomes *see* KEGG
 L50 96, 102–103, 107
 lactose operon 164
 Levenshtein distance 65, 257
 LG50 96
 local sequence alignment 55, 258
 locus 90, 172
 logo 245, 249
 lookup table 62–63
 MAFFT 297–298
 MAG 304
 manifest file 268–269, 274–275
 Maxam, Allan M. 3
 MaxBin 304, 321, 326

- maximum likelihood 261
 Mendel, Gregor 109
 message passing interface *see* MPI
 MetaBAT 304, 326
 Metaeuk 104
 Metagenome-Assembled Genome *see* MAG
 metagenomics 253–254, 265, 301–302, 304, 326
 methylation 213–214
 microscopy 9
 missense 109, 114, 143–145, 148, 150
 molecular clock 261
 motif 215–217, 225, 243–250
 MPI 93
 mpileup 79, 84, 114–116, 121, 123
 MSA 146
 multiple sequence alignment *see* MSA
 multiplexing 5, 254, 270, 303–304, 306
 N50 96, 102–103, 107
 naïve Bayes 289, 291–293, 304
 nanopore sequencing 9, 90
 National Center for Biotechnology Information 50
 Neanderthal 49
 Needleman-Wunsch 55, 259
 negative binomial 178–180, 185–189
 NEWICK 298
 NJ 177, 257, 261
 nodes 56–57, 91–93
 nonsense 110, 144, 148, 150
 nonsynonymous codon 109, 146
 normalization 174, 176, 182–183, 186–187
 novoalign 65
 nucleosome 213, 216
 oligo 6, 8, 249
 ONT 9–10, 90, 203, 239–240, 305
 open reading frame *see* ORF
 open-reference clustering 255, 278, 280, 282, 289,
 301
 operon 2, 164
 ORF 2, 164, 325
 ortholog 103–105, 108
 overlap 81, 89–93, 107
 overlap-consensus 90–91, 107
 Oxford Nanopore Technologies *see* ONT
 PacBio 9–10
 Pacific Bioscience *see* PacBio
 PairHMM 126
 pairwise alignment 90–91, 107, 126, 259
 Pandas 104
 parsimony 261
 PCoA 300
 peak calling 215–217, 223
 phosphorylation 213
 phred 12–14, 23–24
 PICARD 79, 86, 130, 132, 134–135, 210
 Pielou's Evenness 262
 plants 49, 253
 polymerase chain reaction 4
 position-specific scoring matrix *see* PSSM
 prefix tree 56
 principal coordinate analysis *see* PCoA
 prodigal 104, 323, 325–326
 PSI-BLAST 146
 PSSM 146
 purine 1, 109, 247
 pyrimidine 1, 109, 247
 pyrophosphate 5–6
 pyrosequencing 5–6
 Q score 12–13
 quasi-negative binomial 178–179, 196
 QUAST 100–103, 107–108, 317–318, 326
 rank table 62–63
 RDP 258–259
 RefSeq 50, 228, 311
 regular expression 81
 representative sequence 255, 258–261
 ribonucleic acid *see* RNA
 Ribosomal Database Project 258–259, 302
 ribosome 2, 254
 richness 261–262, 299
 RIN 18
 RNA integrity *see* RIN
 RNA-Seq 163, 165–169
 Robert Holley 3
 Roche 454 5–7
 RPKM 174–175, 210, 212
 Samtools 53–54
 Sanger, Frederick 3
 Sanger sequencing 4, 90, 254
 SARS-CoV-2 97, 99, 116, 118, 127, 155–157, 159–160
 scaffold 89, 94–100, 102–103, 304–305, 316–320
 seed-and-extend 65
 seeding 73
 SEPP 104
 sequence alignment 53, 55, 63, 65, 70, 87, 90
 sequencing depth 41, 45, 90, 114, 160, 175, 190
 Shannon's diversity index 261–262
 sickle 305, 310, 325
 SIFT 145–148, 152, 161
 singleton 42, 46, 309
 SMRT 9, 90, 98, 166

- SNP 135–136, 139–143
 SnpEff 145, 148–151, 161
 SNV 110, 114, 116
 SOAP 65, 87
 SOLiD 6, 8, 135, 156, 253
 sonication 4, 214
 SPAdes 92, 97–100, 102, 105–108,
 315–316
 splicing site 148, 164
 SRA-toolkit 14–15, 70, 215, 265, 305
 ssDNA 3, 5, 7, 9–10
 STAR 58, 70–71, 76–78
 stop codon 110, 143
 stop-gain 143, 148
 structural variant 113
 suffix array 55, 57–60, 65, 72, 77–78, 86–87
 suffix tree 55–58, 65, 87, 246
 sulfurylase 5
 SV 110
 SW 46, 55, 65, 72–74, 326
 tag 69–70, 80, 134, 215–216
 tandem repeats 89
 TATA box 164
 TGS 8–9, 45, 85
 third-generation sequencing 8
 thymine 1–2, 13, 213, 247
 TPM 175
 transcription start site *see* TSS
 transcripts per million *see* TPM
 transition 109, 256–257
 translocation 109–110
 transversion 109
 trie 11, 56, 65, 87
 trimmomatic 42–47
 triphosphate 4–5
 TruSeq 97
 TSS 144, 164, 233–238
 ubiquitylation 213
 UniFrac 261–263, 297, 300
 untranslated region *see* UTR
 UPGMA 261
 Uracil 1
 USEARCH 255
 UTR 144, 148
 variant call format *see* VCF
 variant calling 113–116
 Variant Quality Score Recalibration *see* VQSR
 VCF 110–113, 115–116
 vertexes 91
 virion 1–2
 virus 99, 116, 253
 volcano 199–200, 208–209
 VQSR 139–141
 VSEARCH 258–259, 278, 280–282
 Watson, James 3
 WPGMA 261