Computational methods for de novo assembly and sequencing error correction of short reads in the era of (viral) metagenomics

Dissertation

for the award of the degree
"Doctor rerum naturalium"
Division of Mathematics and Natural Sciences of the
Georg-August-Universität Göttingen

within the doctoral program
International Max Planck Research School for Genome Science
of the Georg-August University School of Science (GAUSS)

submitted by
Annika Jochheim
from Wolfsburg, Germany

Göttingen, 2022

Thesis Advisory Committee

Dr. Johannes Söding
Research Group Quantitative and Computational Biology, Max Planck Institute for Multidisciplinary Sciences

Prof. Dr. Burkhard Morgenstern
Institute for Microbiology and Genetics, Department of Bioinformatics, Georg-August-University Göttingen

Prof. Dr. Rolf Daniel
Institute for Microbiology and Genetics, Department of Genomic and Applied Microbiology & Göttingen Genomics Laboratory, Georg-August-University Göttingen

Members of the Examination Board

Dr. Johannes Söding (1st reviewer)
Research Group Quantitative and Computational Biology, Max Planck Institute for Multidisciplinary Sciences

Prof. Dr. Burkhard Morgenstern (2nd reviewer)
Institute for Microbiology and Genetics, Department of Bioinformatics, Georg-August-University Göttingen

Further Members of the Examination Board

Prof. Dr. Rolf Daniel
Institute for Microbiology and Genetics, Department of Genomic and Applied Microbiology & Göttingen Genomics Laboratory, Georg-August-University Göttingen

Prof. Dr. Jan de Vries
Institute for Microbiology and Genetics, Department of Applied Bioinformatics, Georg-August-University Göttingen

Prof. Dr. Michael Altenbuchinger
Department of Medical Bioinformatics, University Medical Center Göttingen

Prof. Dr. Anne-Christin Hauschild
Department of Medical Informatics, University Medical Center Göttingen

Date of the oral examination: October 4, 2022

Acknowledgements

This was a long journey - more like a marathon than a sprint - that would not have been possible without the people around me.

First, I would like to thank my supervisor Dr. Johannes Söding for giving me the opportunity for my doctoral research and to work and grow in his group. From our first interview to writing my thesis, your excitement about science has heavily motivated me. I am grateful for the space and freedom you provided to develop and pursue my own ideas. I have learned a lot and grown personally and scientifically. Moreover, I am endlessly grateful for all your support in good, bad and random times and for always having an open door. Further, I thank Prof. Dr. Burkhard Morgenstern and Prof. Dr. Rolf Daniel for being part of my thesis committee and guiding my way, providing valuable comments and suggestions during this journey. I am thankful to Prof. Dr. Jan de Vries, Prof. Dr. Michael Altenbuchinger and Prof. Dr. Anne-Christin Hauschild for taking their time and being part of my examination board. I also wish to thank the members of my graduate school, and everyone involved, who enabled my doctoral research. I would like to thank Dr. Henriette Irmer and Frauke Bergmann for their assistance in all administrative tasks.

A special thanks goes to Prof. Dr. Martin Steinegger for his co-supervision. Thank you for taking the time for our discussions no matter what time of day, no matter what time zone, you have always had an open ear. I enjoyed working with you and have greatly benefited from your expertise and support. I would also like to thank all present and former members of the Söding lab. Especially, I thank Dr. Eli Levy Karin and Dr. Clovis Galiez for their guidance and advice at the beginning of my graduate career, and Dr. Milot Mirdita for all scientific discussions and for always volunteering technical support. Further, I am grateful to Étienne Morice for the mathematical support in my project. I thank Dr. Salma Sohrabi-Jahromi and Michel van Kempen for being caring office neighbors, as well as Louis Kraft, Anton Farr and Jakub Wojciechowski for trusting me as their supervisor and giving me the opportunity to grow as a mentor. I am also grateful to Hannah Rauterberg, Florian Jochheim, Dr. Wanwan Ge, Dr. Yazhini A and Dr. Gesa Werner for proofreading this work.

I thank my parents and my brother for always supporting me, believing in me and encouraging me. You have always given me a lot of strength to reach my goals. Finally, I would like to thank Florian, for accompanying me on this journey since the very beginning. You always supported me, personally and scientifically. Especially when I needed it the most, you were always at my side, never left it. My gratitude to you is hard to put into words. I could not have completed this work without your support. Thank you for believing in me.

Abstract

Viruses can affect all types of living cells, including bacteria, archaea and eukaryotes. Especially in the form of bacteriophages - bacteria infecting viruses - they have a huge impact on their host communities, driving bacterial diversity, shaping composition, interactions, functions and even genomes. Despite their importance, only very little is known about the viral component in microbial communities. Recent advances in sequencing technologies and the advent of metagenomics allow for a culture-independent analysis of the whole genetic material from an environmental sample. This allows to discover previously uncharacterized and newly emerging viruses within their natural environment. In this work, I address two computational tasks in the data analysis in metagenomics, with a focus on the viral fraction.

In the first part of this thesis, I introduce PenguiN (protein-guided nucleotide assembler), a new metagenomic de novo assembler. PenguiN utilizes full-read overlaps calculated in linear time on both amino acid and nucleotide sequences within a greedy iterative assembly procedure. PenguiN is built upon the protein-level assembler Plass. In a first stage, six-frame translated reads are assembled to proteins, whereas the underlying nucleotide sequence is assembled simultaneously, resulting in full open reading frames (ORFs). In a second stage, the resulting ORFs are then linked with nucleotide reads to bridge intergenic regions as well, enabling the assembly of whole genomes. Additionally, I introduce a new extension strategy using a Bayesian model to identify the best overlaps in each iteration and describe a strategy to detect circular sequences.

Utilizing full-read overlaps in linear time, PenguiN overcomes the sensitivity-specificity trade-off seen in k-mer based (de Bruijn graph) state-of-the-art metagenomic assemblers, while being much faster than existing overlap-based assemblers. Moreover, focusing on the viral fraction of microbial communities, I show that PenguiN can assemble longer contigs and more complete genomes than existing assembly tools and overcomes the typical loss of population diversity seen in metagenomic assemblies. Further, I show that PenguiN can also obtain long viral contigs at very low read coverage. On a simulated metagenome, I obtain a 3- to 11-fold increase in the per-nucleotide sensitivity compared to the next best tool at comparable per-nucleotide precision. On a metatranscriptomic dataset from 82 aquatic and activated sludge samples, PenguiN assembles about 75-90% (343-376) more complete ssRNA phage genomes than state-of-the-art tools.

In the second part of the thesis, I introduce CoCo, a new software tool for sequencing error correction. By identifying sequencing errors as discontinuities in spaced k-mer frequencies along a read, CoCo can make local decisions instead of using a global threshold. Together with a very conservative two-side correction strategy, this allows to be more specific for low frequency variants than tools that rely on global k-mer count statistics. Moreover, I introduce a memory efficient data structure to store the spaced k-mer counts. This makes it possible to run CoCo on large and complex metagenomic datasets. Using CoCo's corrected sequencing reads for PenguiN's assembly improves the final contigs, which become more continuous and more accurate.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANI	average nucleotide identity
ASV	amplicon sequence variant
DNA	deoxyribonucleic acid
dsDNA	double-stranded DNA
dsRNA	double-stranded RNA
FN	false negative
FP	false positive
GWDG	Gesellschaft für Wissenschaftliche Datenverarbeitung Göttingen
HIV	human immunodeficiency virus
HMM	hidden Markov model
HPC	high-performance computing
HRV	human rhinovirus
ICTV	International Committee on Taxonomy of Viruses
MAG	metagenomic assembled genome
MSA	multiple sequence alignment
NGS	next-generation sequencing
OLC	overlap-layout-consensus
ONT	Oxford Nanopore Technologies
ORF	open reading frame
OTU	operational taxonomic unit
PCR	polymerase chain reaction
RAM	random-access memory
RAR	repeat-genome-repeat structure
RNA	ribonucleic acid

SMRT single molecule real-time

SNP single nucleotide polymorphism

ssDNA single-stranded DNA

ssRNA single-stranded RNA

SSE Streaming SIMD Extensions

TN true negative

TP true positive

1. Introduction

Viruses are the most abundant entities on earth [Breitbart and Rohwer, 2005; Mokili et al., 2012]. It is estimated that around 10^7 virus particles occur in one milliliter of seawater [Bergh et al., 1989], up to 10^9 in a gram of soil [Williamson et al., 2017] and that the human body harbors more than 10^{15} virus particles, meaning a 10-fold increase to bacteria and a 100-fold increase in comparison to the number of human cells [Mokili et al., 2012; Savage, 1977]. Most of them are bacteria-infecting viruses, so-called bacteriophages (short: phages) (reviewed in [Guerin and Hill, 2020; Shkoporov and Hill, 2019]). Despite their small size, they have enormous impact on their host community and ecosystems: They play key roles to maintain a stable and balanced gut microbiome and have therefore significant impact on human health [Manrique et al., 2016; Moreno-Gallego et al., 2019; Norman et al., 2015; Sutton and Hill, 2019]. They impact carbon and nutrient cycling in terrestrial systems [Albright et al., 2022; Emerson et al., 2018; Trubl et al., 2018; Williamson et al., 2017] and marine systems [Danovaro et al., 2011; Fuhrman, 1999; Suttle, 2005, 2007; Wilhelm and Suttle, 1999; Zimmerman et al., 2020] and are in total responsible for 20-40% of bacterial lysis in the ocean per day [Brum and Sullivan, 2015; Keen, 2015; Suttle, 2007]. Furthermore, they can facilitate horizontal gene transfer [Colavecchio et al., 2017; Frazão et al., 2019; Gyles and Boerlin, 2014; Irwin et al., 2022; Moon et al., 2020; Sano et al., 2004; Zeidner et al., 2005] and act therefore as essential evolutionary drivers, increasing the "genetic space" and enabling their hosts to adapt to environmental changes by supplementing existing or providing entirely novel functionality [Boucher et al., 2003; Frazão et al., 2019; Gogarten and Townsend, 2005; Irwin et al., 2022; Williams, 2013]. Incorporating viruses is therefore crucial for our understanding of microbial communities, their structure, and function, as well as our understanding of biological and biogeochemical processes in many environmental systems.

Historically, the study of viruses relies on isolation and co-culturing and was mainly focused on viruses causing specific diseases [Miller et al., 2013; Mokili et al., 2012; Simmonds et al., 2017]. This heavily limits our knowledge to pathogenic viruses. However, recent advances of shotgun sequencing and the advent of metagenomics allow studying viruses more broadly. Studying viruses from metagenomic samples reveals many previously unknown viruses - especially phages - and hint on a huge diversity, which still remains to discovered [Breitbart et al., 2002; Gregory et al., 2020, 2019; Reyes et al., 2010; Shkoporov et al., 2019]. My work has addressed two main problems in the computational analysis of metagenomic sequencing data: (1) de novo assembly and (2) sequencing error correction, with a main focus on the viral fraction. Before going into the details of these methods, I provide the reader in the remaining part of this chapter with the

necessary information about the diversity of viruses and viral genomes (section 1.1), sequencing technologies (section 1.2) and the advent of microbial and viral metagenomics (section 1.3). This is followed by a comprehensive review of different assembly approaches (section 1.4) and sequencing error correction approaches (section 1.5). Thereby, I point out current challenges and limitations with these methods and address the relevance for the new methods introduced in this thesis. Finally, I summarize the aims of this thesis and outline the scope of the following chapters (section 1.6).

1.1. The diverse universe of viruses

Viruses are small infectious particles (mostly 20–300 nm) comprised of genetic material, deoxyribonucleic acid (DNA) or ribonucleic acid (RNA) within a protein shell, a so-called capsid. All viruses have in common that they lack ribosomes, instead they hijack the host machinery for their replication. Besides these common features, viruses are extremely diverse, both in morphology and genome structure [Louten, 2016; Sanjuán and Domingo-Calap, 2021]. Their genome can be comprised of DNA or RNA, which can either be double-stranded or single-stranded (ssDNA, dsDNA, ssRNA, dsRNA). Additionally, single-stranded RNA viruses can be either positive sense or negative sense, depending on its genome's orientation relative to the viral messenger RNA (mRNA). Furthermore, the structure of the genome can either be linear or circular, and it can be made up of a single nucleic acid molecule or is divided into multiple segments [Marintcheva, 2018]. Thus, viruses already exhibit a plethora of variation. Also, the size of the genome varies significantly. Typically, the size of viral genomes range from 7-20 kbp [Louten, 2016]. However, there are circoviruses, with only two to three genes and a genome size of 1.7-2.1 kbp [Breitbart et al., 2017], and giant viruses including mimiviruses with sizes of >1 Mbp [Raoult et al., 2004] or the even more extreme pandoraviruses [Pereira Andrade et al., 2019], with up to 2.5 Mbp [Philippe et al., 2013]. The range of phage genome sizes is a bit shorter, but still ranges from 2.4 kbp to megaphages with genomes >540 kbp (see Fig. 1.1). Generally, RNA viruses are shorter than DNA viruses [Chaitanya, 2019] and evolve more quickly. They have high mutation rates of $10^{-6} - 10^{-4}$ substitutions per nucleotide per cell infection, whereas those from DNA viruses range from 10^{-8} to 10^{-6} [Sanjuán and Domingo-Calap, 2016]. This is in part driven by the fact that RNA viral replication is not influenced by the host's DNA repair mechanism [Sanjuán and Domingo-Calap, 2021]. Other commonly observed patterns are that single-stranded viruses show higher mutation rates than double-stranded viruses, and smaller viral genomes tend to mutate faster than those with larger genomes [Sanjuán and Domingo-Calap, 2016]. This is in accordance with the general inverse correlation of mutation rate and genome size also seen for cellular microorganisms [Lynch, 2010].

The high mutability of viruses is thereby driven by the error-prone replication of viruses due to a highly error-prone polymerases [Dolan et al., 2018]. Additionally, RNA virus replication lack proofreading mechanisms (except coronaviruses) [Smith et al., 2013] and certain DNA viruses have developed strategies to avoid or interrupt the DNA repair mechanism from the host cell

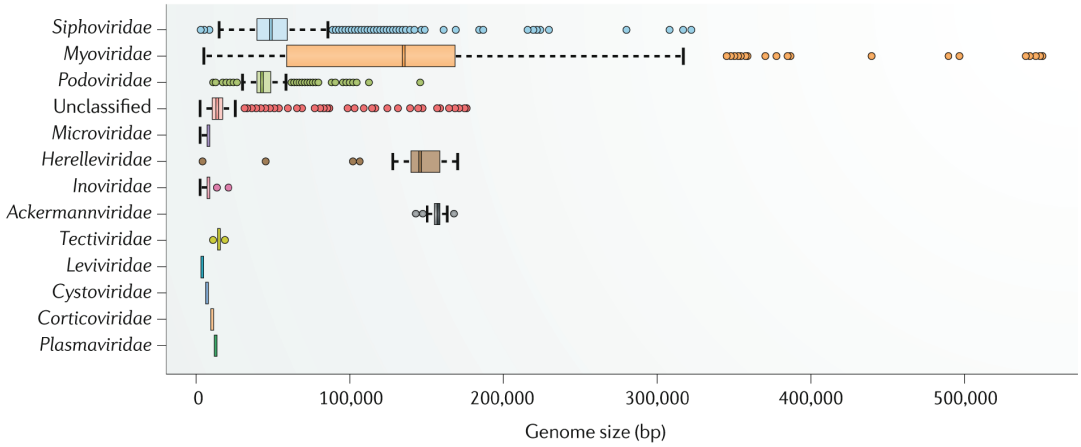


Figure 1.1.: Distribution of phage genome sizes. The Figure is taken from [Dion et al., 2020].

[Pereira-Gómez and Sanjuán, 2015]. Next to the nucleotide mutations, recombinations are a further driving factor of viral diversity. They occur upon coinfection due to reassortment events in segmented viruses or template switching [Pérez-Losada et al., 2015; Sanjuán and Domingo-Calap, 2021]. In total, the high mutability leads to high genetic diversity, which plays a key role in evolutionary processes. It allows viruses to adapt to environmental changes, escape host immune responses, uphold pathogenesis and to evolve drug resistance [Sanjuán and Domingo-Calap, 2021]. The competition with other viral members of their community thereby provides selective pressure and determines, together with stochastic processes such as genetic drift, how any new strain will spread [Bell, 2021; Retel et al., 2019].

Despite the enormous variation, there is effort for the classification of viruses. Historically, viruses were classified based on their morphology and host range. In 1971, the non-hierarchical Baltimore classification system was proposed [Baltimore, 1971]. It groups viruses based on their nucleic acid genome type and replication strategies into seven classes: dsDNA viruses, ssDNA viruses, dsRNA viruses, (+)ssRNA viruses, (-)ssRNA viruses, (+)ssRNA viruses using reverse transcription replication, and dsDNA viruses using reverse transcription replication (added later). Even though the system is not "official" and does not reflect phylogenetic relationships, it is still widely used [Koonin et al., 2021]. To provide an official classification of viruses, which adopts a hierarchical system to reflect phylogenetic distances, the International Committee on Taxonomy of Viruses (ICTV) was founded in 1966 [Lefkowitz et al., 2018]. Their classification previously described five ranks, but has been expanded to 15 in 2019 [ICTV, 2020]. The classification is thereby based on genome structure and organization, but also on sequence-based analysis [Simmonds, 2015]. As virus discovery continues, especially due to the newest advance in viral metagenomics, the system is regularly updated to incorporate the newest proposal. The most recent update was published in 2021 [Walker et al., 2021]. It currently describes 6 realms, 10 kingdoms, 17 phyla, 2 subphyla, 39 classes, 65 orders, 8 suborders, 233 families, 168 subfamilies, 2606 genera, 84 subgenera, 10 434 species [Walker et al., 2021].

Among the viruses, phages are thought to be the most diverse group, with approximately ten times more diversity than their hosts [Dion et al., 2020]. Especially (viral) metagenomics has thereby allowed to estimate this richness and give insights into the extreme diversity of phages in numerous environments [Dion et al., 2020]. Moreover, it hints at the bias of previous methods. Whereas the (larger) double-stranded DNA tailed phages account for the majority of phages described previously through the use of electron microscopy and culture-based methods, current studies increasingly identify ssDNA [Hopkins et al., 2014; Roux et al., 2019b] and ssRNA phages [Callanan et al., 2020]. In a recent review, Mirzaei et al. suggest that many phages have been previously overlooked and a lot of phage diversity has yet to be discovered [Mirzaei et al., 2021].

1.2. Revolution of sequencing technologies

The ability to determine the genetic sequence of an organism was an enormous breakthrough and has changed our understanding of biology. Over the last decades, several different sequencing technologies were developed and constantly enabled to obtain the nucleotide sequence of DNA or RNA molecules at an ever-increasing speed and decreasing costs. The enormous progress is described in several review papers [Goodwin et al., 2016; Heather and Chain, 2016; Kchouk et al., 2017; Liu et al., 2012; Reuter et al., 2015; Slatko et al., 2018] and briefly summarized in the following. Whereas DNA fragments can be sequenced directly, RNA sequencing usually requires the reverse transcription into cDNA first [Stark et al., 2019]. However, then the same technologies can be applied.

DNA sequencing goes back to the mid of 1970s (first-generation sequencing), when Maxam and Gilbert developed a technology where radioactively labeled DNA molecules are chemically cleaved at specific bases, followed by an electrophoresis, which sorts the cleaved DNA fragments by lengths and allows therefore to infer the nucleotide sequence [Maxam and Gilbert, 1977, 1980]. Around the same time, Sanger developed a method to determine the nucleotide sequences using chain termination [Sanger and Coulson, 1975; Sanger et al., 1977]. Thereby, the DNA is synthesized on the template strand via DNA polymerization reactions in the presence of dideoxynucleotides (ddNTPs), modified dNTPs that lack a 3' hydroxyl group. If a ddNTP is incorporated, no phosphodiester bond can be established by the DNA polymerase and the chain process terminates. This leads to fragments of different sizes, which can be separated using gel electrophoresis and subsequently visualized via autoradiography. The radioactively labelled ddNTPs were later replaced by fluorescently labeled ddNTPs [Smith et al., 1986] and a capillary electrophoresis system was introduced to separate the fluorescently labeled DNA fragments [Swerdlow and Gesteland, 1990]. This improved speed and allowed for the automatization of the DNA sequence analysis [Hunkapiller et al., 1991]. Sanger sequencing was first commercialized by Applied Biosystems (ABI, now part of ThermoFisher) and became widely used for many years [Kchouk et al., 2017]. Today, it still remains in use in clinical settings due to its simplicity and high accuracy (99.99%), cost-efficiency for small numbers of targets as well as the missing approval by the FDA for many newer sequencing technologies.

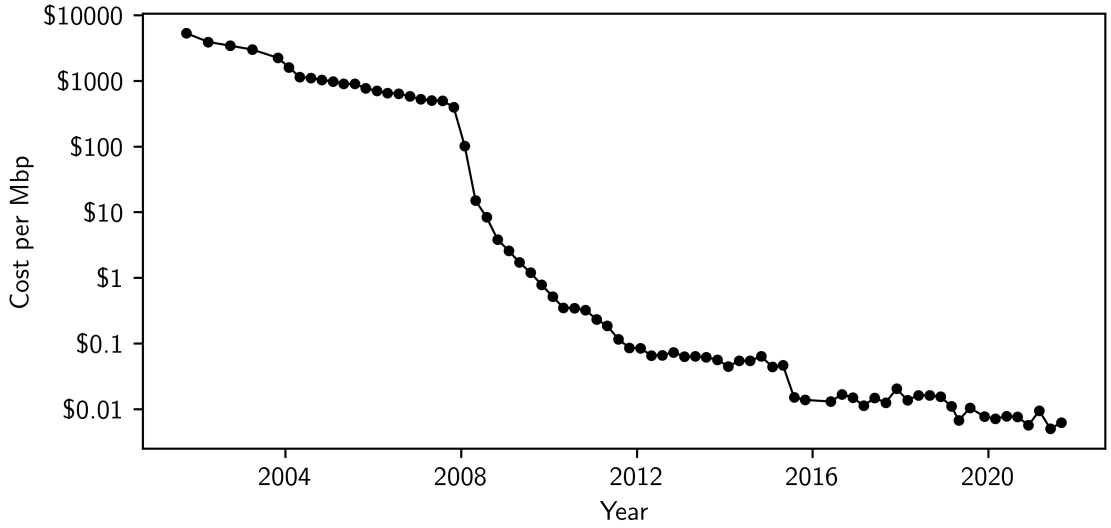


Figure 1.2.: Sequencing cost per megabase of DNA sequence as reported by the National Human Genome Research Institute (NHGRI). The enormous drop in sequencing cost allows for the advent of data-heavy research areas such as metagenomics. The numbers were obtained from genome.gov/sequencingcosts.

In 2001, the first human genome (totaling $\sim$3 billion bp) was published using Sanger sequencing technology [Venter et al., 2001]. However, it was a $\sim$2.7 billion US$ effort and over 13 years of work, with approximately 10 months spent on the sequencing itself (September 1999 - June 2000) [Venter et al., 2001]. Since then, tremendous progress has been made in genome sequencing technologies. Starting from 2005, a number of next-generation sequencing (NGS) technologies arose: First, pyrosequencing [Margulies et al., 2005; Nyrén et al., 1993], which was licensed to 454 Life Science (later Roche), followed by Illumina dye sequencing (Solexa) [Bentley et al., 2008] and SOLiD (Supported Oligonucleotide Ligation and Detection) sequencing by Life Technologies (ABI/SOLiD) [Shendure and Ji, 2008]. All these high throughput sequencing (HTS) technologies are characterized through massive parallel sequencing that allow to generate millions to billions of reads in a single sequencing run with a higher speed and lower cost in comparison to first generation sequencing [Kchouk et al., 2017]. In the year 2008 it was possible to sequence the full human genome within two months for US$1 million [Wheeler et al., 2008] and 2014 Illumina claimed to reach the US$1000 genome [Check Hayden, 2014]. The massive drop in sequencing costs for a megabase of DNA continued further (see Fig. 1.2) and enabled several new research areas such as metagenomics [Desai et al., 2012], transcriptomics [Wang et al., 2009] and epigenomics [Clark et al., 2016].

Currently, the most prevalent sequencing technology is Illumina sequencing. It applies a sequencing by synthesis approach using reversible terminators [Bentley et al., 2008]. Prior to the sequencing, DNA is randomly fragmented, denatured into single strand fragments, ligated to adapters and annealed to complementary oligonucleotides on the surface of a flow cell. Then a PCR bridge amplification is performed resulting in approximately 1 million copies of the original fragment, so-called clusters, making the signal detectable. Linearized single-strand DNA

fragments act then as templates for the sequencing by synthesis. In cycles, a DNA polymerase synthesizes the complementary strand using flurophore-labelled, terminally blocked nucleotides, starting from a primer. In each cycle (1) a single nucleotide is incorporated in each fragment, (2) remaining nucleotides are washed away, and (3) each cluster is excited by a laser; thus the fluorescent signal is recorded to identify the incorporated base. (4) Afterwards, the terminator with the fluorescent label is removed, reactivating the blocked 3' hydroxyl group, and allowing for the incorporation of the next nucleotide in a new cycle. The number of performed cycles determines thereby the length of the sequencing read. The first Illumina/Solexa sequencer *Genome Analyzer* could produce reads of ~ 35 bp. Since then, Illumina launched several machines with a variety of read length and throughput [Slatko et al., 2018]. Currently, *MiSeq* platforms produce 1-25 million reads per run with a length of 2×300 bp, *HiSeq* platforms generate 150 million to 4 billion reads of 2×150 bp and *NovaSeq* can produce 800 million up to 20 billion reads of 2×150 bp. All the Illumina platforms are capable of paired-end sequencing, the sequencing of a fragment from both sides with a known distance (insert size), and commonly show very low error rates ($< 0.1 - 0.6\%$), with substitution errors dominating and accumulating especially towards the end of the reads [Stoler and Nekrutenko, 2021].

Also, new technologies have been developed that produce much longer reads (third-generation sequencing). In 2011, Pacific Biosciences (PacBio) released a sequencer that uses single molecule real time (SMRT) sequencing [Eid et al., 2009] and then, in 2014, the first nanopore sequencer from Oxford Nanopore Technologies (ONT) became available [Clarke et al., 2009; Manrao et al., 2012]. While relying on fundamentally different principles, both technologies have in common that they do not require DNA amplification, provide data in real-time and produce very long reads, ranging commonly from 10-30 kbp [Amarasinghe et al., 2020; Athanasopoulou et al., 2021; Van Dijk et al., 2018], with records up to a few megabasepairs [Payne et al., 2019]. However, they are much more expensive, require larger amount of starting material and have significantly higher error rates than previous technologies, $\sim 15\%$ for SMRT in single pass sequencing and $\sim$ 3-15% for nanopore sequencing [Dohm et al., 2020; Van Dijk et al., 2018; Zhang et al., 2020].

At the time of writing, Illumina short-read sequencing platforms are still the most widely used, with Illumina owning 80% of the sequencing market. However, if the third-generation sequencing technologies improve further, they probably have the potential to revolutionize the field of genomics and metagenomics [Van Dijk et al., 2018].

1.3. Microbial and viral metagenomics

A major drawback in the study of microbial communities is that as many as 99% of the microorganisms cannot be cultivated in the lab [Schloss and Handelsman, 2005]. As viruses depend on a cellular host for replication, this also has heavily limited investigations of the complex world of viruses until very recently [Breitbart and Rohwer, 2005; Roux et al., 2021]. Culture-based approaches using for example the traditional 'plaque' assays, in which viruses are isolated from a

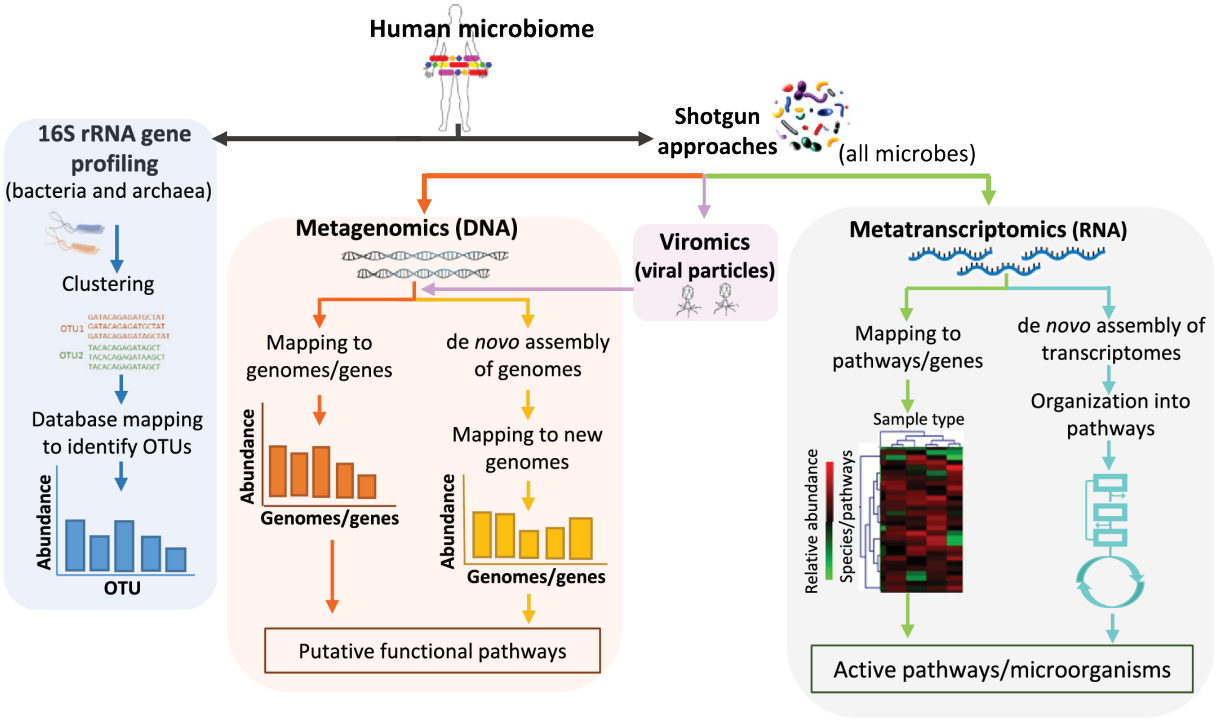


Figure 1.3.: Different culture-independent strategies for the analysis of microbial communities. The 16S rRNA gene profiling can be applied to bacteria and archaea, it aims for the taxonomic profiling. Sequencing reads are either clustered into operational taxonomic units (OTUs) based on sequence similarity (usually 97%) or denoised to amplicon sequence variants (ASVs). Then, taxonomy is assigned by comparing the representative sequences of the OTUs or ASVs against reference databases. In contrast, the shotgun sequencing approaches can be applied to the whole community, leading to an unbiased view and more in-depth understanding. Metagenomics thereby describes the study of the whole genetic material. After sequencing, the reads can be mapped to reference genomes/genes or assembled to longer fragments and ideally whole genomes. This approach provides much more information than the study of a single marker gene. In contrast to metagenomic, metatranscriptomic focuses on the active genes and expressed transcripts in the microbiome at a certain time point. It allows obtaining gene expression profiles and to identify active pathways based on RNA sequences. Viral metagenomics (viromics) combines technically metagenomics and metatranscriptomics, as viruses can be either made from DNA or RNA. Before sequencing, viral particles are usually enriched. Then viromics analysis follows the typical metagenomic workflow. Figure taken from [Bikel et al., 2015].

bacterial host grown in liquid culture, could only be performed for a minor fraction, and the vast majority remained unknown [Breitbart and Rohwer, 2005]. Nowadays, the study of viruses can be carried out using metagenomic-based strategies that do not rely on isolation and cultivation [Dávila-Ramos et al., 2019].

The idea to study the complexity of natural microbial populations directly from environmental samples was first proposed by Pace and colleagues in 1986 [Pace et al., 1986]. As sequencing was costly and time-consuming at that time, they focused on single ribosomal RNA (rRNA) sequences as phylogenetic marker genes. For populations of limited complexity they isolated 5S rRNAs, for more complex populations they performed shotgun cloning of 16S rRNA genes using purified DNA, sequenced them with 16S rRNA-specific primers and subsequently compared them to existing 16S rRNA sequence collections to infer phylogenetic relations [Olsen et al., 1986; Pace et al., 1986]. The idea of using ribosomal RNA sequences as molecular markers for

the phylogenetic analysis was already proposed a few years earlier by Woese and Fox [Woese and Fox, 1977]. With the advent of polymerase chain reaction (PCR) it became possible to selectively amplify, and subsequently sequence, the rRNA gene sequences (amplicon sequencing). Today, the most widely used marker gene for bacteria and archaea is the 16S rRNA gene [Johnson et al., 2019; Langille et al., 2013; Tringe and Hugenholtz, 2008; Turnbaugh et al., 2009]. The 16S rRNA is the smaller subunit of the ribosome RNA molecule (rRNA) in bacteria and archaea, and its ~ 1500 bp long gene comprises nine hypervariable regions (V1-V9) separated by highly conserved regions. The conserved regions allow for targeting the gene with "universal" PCR primers, whereas the hypervariable regions are used for identification and taxonomic classification [Breitwieser et al., 2019; Johnson et al., 2019]. See Fig. 1.3 for a short overview of a typical pipeline. This approach became widely used and has been the gold-standard to analyze the composition of microbial communities (taxonomic profiling) for decades [Hugenholtz et al., 1998; Hugenholtz and Pace, 1996; Johnson et al., 2019; Lozupone and Knight, 2007; Matsuo et al., 2021; Pace, 1997; Ward et al., 1990]. Analogous to the 16s rRNA gene, the 18S rRNA gene is used for eukaryotes [Kounosu et al., 2019; Tanaka et al., 2014] and internal transcribed spacer (ITS) regions for fungi [Ihrmark et al., 2012; Schoch et al., 2012]. However, viruses lack of a universal marker gene, highly limiting the amplicon-based approach for the study of viruses from environmental samples [Breitbart and Rohwer, 2005]. Conserved genes occur only among certain groups of viruses. Some studies investigated for example the diversity of T4-like bacteriophages using the gene 23 (g23), which encodes the major capsid protein in all T4-like phages [Filée et al., 2005; Uyaguari-Diaz et al., 2016]. Other studies examined T7-like Podophages using DNA polymerase genes, [Breitbart et al., 2004] or studied cyanophages using the viral capsid assembly protein gene g20 [Dorigo et al., 2004; Zhong et al., 2002]. Besides the limitation to only a single gene, the major drawback of these approaches is that they are unable to discover novel groups of viruses.

When sequencing cost decreased, it became possible to sequence the whole genetic material from a sample instead of focusing only on single marker genes. The so-called whole metagenomic shotgun sequencing arose. The term "metagenome" was first coined by Jo Handelsmann in 1998 to describe the collective genomes obtained from soil microflora [Handelsman et al., 1998]. Since then, metagenomics have been applied for a range of environments. In contrast to amplicon sequencing, whole metagenomic shotgun sequencing goes beyond taxonomic profiling based on a single gene. It allows recovering whole genome sequences and to retrieve the functional potential of microbial communities [Handelsman, 2004; Simon and Daniel, 2011], see Fig. 1.3. In addition, it can be used to analyze viral-microbial host interactions [Schulz et al., 2020]. In total, metagenomics allows for a more in-depth understanding and provides an unbiased view on the whole microbial community including bacteria, archaea, single-celled eukaryotes and viruses.

Since the study by Breitbart et al. in 2002 on two uncultured marine viral communities [Breitbart et al., 2002], which is often mentioned as the first application of viral metagenomics [Hayes et al., 2017; Liang and Bushman, 2021; Mokili et al., 2012], the number of viral metagenomic studies massively increased. This includes studies of a range of different environments, such as soil

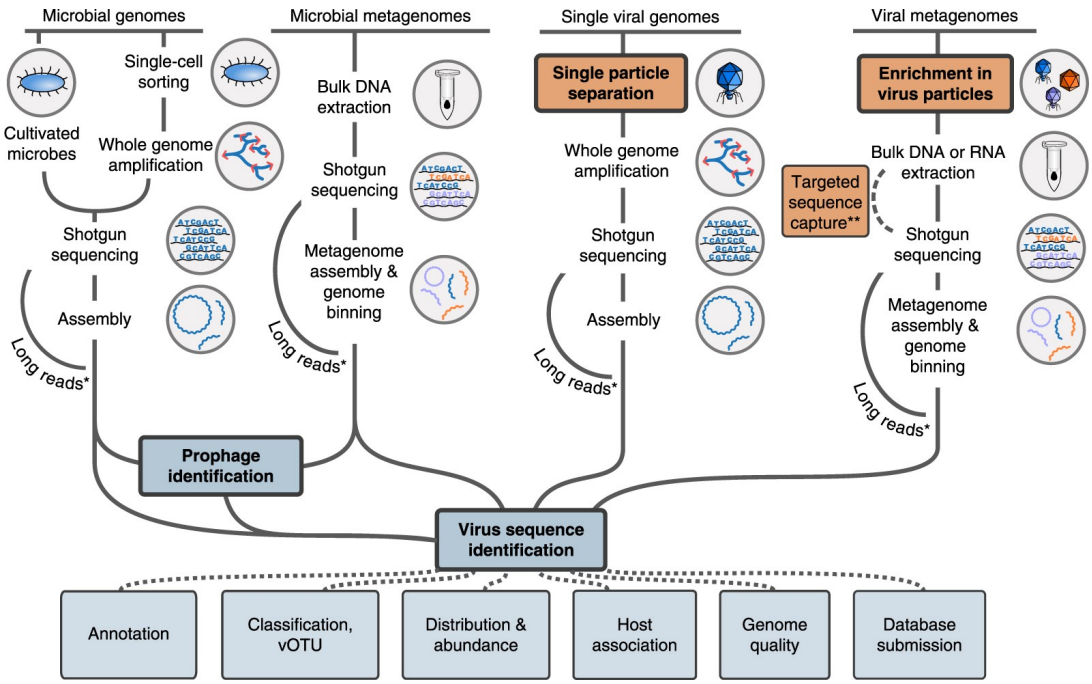


Figure 1.4.: Four sources to identify uncultivated virus genomes. Viral sequences can be obtained untargeted from microbial genomes through co-cultivation or single-cell sorting, and from microbial metagenomes through bulk metagenomic sequencing. Virus-specific sequencing is also possible (orange). Viruses can be obtained from single virus methods using flow cytometry or from metagenomes enriched for virus particles. In this thesis, I consider microbial and viral metagenomes as sources. All methods require assembly methods subsequent to short read shotgun sequencing. In principle, the assembly step can be circumvented for short viruses using long-reads (marked with *). However, long-read technologies have their own drawbacks (see main text). Afterwards, viral sequences and depending on the method also prophages are identified with computational methods. Subsequently, bioinformatic analysis allows then for a deeper understanding of the obtained sequences. Figure is taken from [Roux et al., 2019a].

[Adriaenssens et al., 2015; Schulz et al., 2018], marine environments [Breitbart et al., 2007; Coutinho et al., 2017; Gregory et al., 2019; Hwang et al., 2017; Roux et al., 2016] or the human gut [Breitbart et al., 2003; Gregory et al., 2020; Gulyaeva et al., 2022; Minot et al., 2013; Reyes et al., 2010; Shkoporov et al., 2019].

In general, there are two main approaches in these studies how virus genomes are obtained. Virus genomes can be either retrieved from datasets enriched for virus particles (viral metagenomes) or from bulk metagenomes, including both virus particles and microbial cells [Nayfach et al., 2021b; Roux et al., 2021]. Other sources to obtain uncultivated virus genomes are single-virus methods, where individual virus particles are selected using flow-cytometry [Mirzaei et al., 2021]. See Fig. 1.4 for an overview of the different pipelines.

In viral metagenomics (viromics), virus particles are enriched through a combination of filtration and purification steps. This usually includes size-selective filtration, ultracentrifugation and DNase (or RNase) treatments [Kleiner et al., 2015; Thurber et al., 2009]. The filtration allows for the enrichment of extracellular viruses, but it cannot remove cellular sequences completely [Roux et al., 2013] and viral sequence identification through computational methods still remains necessary later on. A critical step in viral metagenomics is also to obtain a sufficient amount

of nucleotide acids for the sequencing. Due to the small biomass of viruses, an amplification step might be necessary. Most common methods are random amplified shotgun library (RASL) [Rohwer et al., 2001], linker-amplified shotgun library (LASL) [Breitbart et al., 2003], and multiple displacement amplification (MDA) [Hutchison et al., 2005]. However, all these methods introduce biases and no quantitative analysis of relative abundances can be performed in the downstream analysis [Roux et al., 2021].

In bulk metagenomic sequencing, no filtration or amplification step is performed. Instead, the whole genetic material (virus particles and microbial cells) is sequenced together and separation of viral and cellular sequences is only later performed through computational methods. This approach circumvents the amplification bias and allows capturing extra- and intracellular viral sequences, including integrated prophages [Nayfach et al., 2021b]. However, it poses a challenge for the computational methods, as viruses typically represent only a minor fraction of all sequences compared to cellular genomes [Roux et al., 2021].

After the shotgun sequencing, metagenomic assembly and binning can be performed. Assembly is the task to merge the short sequencing reads into longer contiguous fragments called contigs and ideally whole genomes. Assembly is also the main subject of this thesis, and the different assembly approaches are described in detail in the next section (section 1.4). Assembling viral sequences from metagenomic data is especially challenging due to the high viral microdiversity and strain variation, as well as the high mosaicism and proportion of genomic repeats [Mirzaei et al., 2021]. Often a large fraction of viral diversity remains undiscovered in short, fragmented contigs or unassembled reads. Thereby, especially rare viruses are often overlooked [Rose et al., 2016; Roux et al., 2017; Sutton et al., 2019; Trubl et al., 2020; Vázquez-Castellanos et al., 2014]. In a typical metagenomic pipeline, binning is performed after the assembly [Quince et al., 2017]. The process of binning is employed to group contigs into genomic bins, representing sets of sequences estimated to originate from the same genome (MAG, metagenomic assembled genome). However, due to the smaller size of viral genomes and the fact that the assessment of a bin in terms of contamination and completeness is much more challenging than for bacteria due to the absence of a universal marker gene, binning is often omitted in viral metagenomics [Roux et al., 2017]. Still, according to the authors of the very recently published binning framework PHAMB, it can help to identify viral genomes from bulk metagenomic datasets [Johansen et al., 2022]. In principle, assembly and binning methods can be circumvented using long reads. However, long read sequencing technologies have their own drawbacks. They usually require magnitudes more DNA than can be extracted from a virome sample and still show high operation cost and error rates [Mirzaei et al., 2021].

Subsequent to the assembly (and sometimes binning) step, further downstream analysis is performed. Foremost, the assembled contigs have to be sorted into those originating from viruses and microbial contigs. This is even necessary if a viral enrichment step was applied prior to the library preparation, as contamination may still exist [Roux et al., 2013]. The identification of viral sequences is thereby not trivial, as most viruses in metagenomic samples share little to no homology to public databases [Gregory et al., 2019]. Furthermore, prophages poses specific

challenges, as it can be difficult to accurately determine the boundaries of prophages in their host's genome [Roux et al., 2019a]. To date, a number of computational methods exists, which try to recover viral sequences based on similarity- and composition-based strategies. These include PhiSpy [Akhter et al., 2012], VirFinder [Ren et al., 2017], Vibrant [Kieft et al., 2020], Virsorter/Virsorter2 [Guo et al., 2021; Roux et al., 2015a] and CheckV [Nayfach et al., 2021a]. Viral sequence identification can then be followed by more detailed sequence analysis, including functional annotation [Hyatt et al., 2010; McNair et al., 2018] and taxonomic classification [Bin Jang et al., 2019; Eddy et al., 1995], host prediction [Mihara et al., 2016; Villarroel et al., 2016; Zhang et al., 2021], relative abundance estimation [Palermo et al., 2019] and genome quality assessment [Nayfach et al., 2021a].

1.4. Metagenomic assembly from short sequencing reads

The advent of NGS and metagenomic leads to an ever-increasing amount of sequencing data. However, the information within short reads is limited. Thus, it is necessary to merge the reads into longer continuous fragments (contigs), which provide more continuous information about genes, gene complexes and ideally whole genomes. This reconstruction process from the reads is called assembly. The assembly problem for single organisms is already well studied, however metagenomic sequencing data poses new challenges (reviewed in [Ghurye et al., 2016; Lapidus and Korobeynikov, 2021]). (1) The mixture of genomes means that not all reads should be assembled to a single genome, but rather, an unknown number of genomes should be reconstructed. (2) The coverage is highly uneven, resulting from the widely different abundances. (3) Metagenomic samples usually contain a mixture of closely related genomes (population diversity), making it difficult for assemblers to tell them apart. (4) The coverage for the low abundant genomes in a sample is often insufficient. Many assumptions that can be made for single genome assembly (e.g. regarding the coverage of repetitive regions and sequencing errors) do not hold any longer and the use of single genome assemblers is highly limited for metagenomic data [Namiki et al., 2012]. In recent years, many metagenomic assemblers were developed, adopting the concepts from single genome assemblers to the challenges of metagenomic samples. For example, the metagenomic assembler MetaVelvet [Namiki et al., 2012] is based on the single genome assembler Velvet [Zerbino and Birney, 2008; Zerbino et al., 2009] and metaSPAdes [Nurk et al., 2017] extends SPAdes [Bankevich et al., 2012]. The underlying assembly strategies of the metagenomic assemblers are thereby the same as those from the single genome assemblers, respectively.

According to several review papers, including [Ayling et al., 2020; Ghurye et al., 2016; Pop, 2009], the first rough distinction that can be made is reference-based assembly methods (comparative assemblers) and the methods that do not rely on any references (de novo assemblers). In reference-based methods, the reads are mapped against the reference genome and then grouped based on their placement on the reference, which guides the reconstruction of contigs. This approach can be used to target different strains and is implemented for example in the assembler

MetaCompass [Cepeda et al., 2017]. However, as this approach relies on the presence of suitable reference sequences, it is highly limited for metagenomic studies and is omitted in the following. In contrast, de novo assemblers aim to resolve the assembly problem without prior knowledge by reconstructing the contigs directly from the reads. Consequently, they are much more suitable for (viral) metagenomic studies where up to 90% of the sequences do not share any homology to reference databases [Gregory et al., 2019]. However, the task of de novo assembly has been proven to be NP-hard [Medvedev et al., 2007]. One heuristic approach to solving this is to assemble with a greedy strategy, employed in the earliest assemblers [Ghurye et al., 2016]. Currently, two classes of algorithms are widely used overlap-layout-consensus (OLC) assembly and de Bruijn graph (dBg) assembly approaches, which both correspond to the idea that the assembly problem can be described as a graph problem [Miller et al., 2010]. In the following, I describe the concepts behind these three de novo approaches in more detail. Notorious, the greedy assembly approach is thereby a more general strategy and can be used graph-free [Steinegger et al., 2019] but also based on graph structures [Schmidt et al., 2009].

1.4.1. Greedy assembly strategy

The greedy assembly strategy describes an intuitive process where reads are merged in an iterative manner by always choosing the best overlapping reads for extension. The decision for the best overlap depends on the implementation, e.g. the sequence identity of the overlap region, but is always made locally. The assembly process stops when no more reads or contigs can be joined. Due to the local optimization, the methods employing the greedy approach can get blocked or can result in misassemblies when multiple equally likely extension are possible, e.g. within repetitive sequences [Pop, 2009]. However, it preserves the co-occurrence of mutations within one read and can benefit from long overlaps to accurately distinguish closely related sequences (see own results in section 2.5.2). Usually, sequencing errors do not affect the strategy, as they can be considered by the function to be optimized [Ghurye et al., 2016]. The greedy assembly strategy was mainly employed by early single genome assemblers for Sanger sequencing data, such as TIGR [Sutton et al., 1995] or Phrap [Green, 1996]. However, it is not a popular approach in today's metagenomic studies as it is limited by the all-against-all read overlap computation, which typically scales quadratically in runtime with the number of reads.

Recently, our group developed the protein assembler Plass [Steinegger et al., 2019], which overcomes this limitation by implementing a linear time approach for the overlap computation. As a result, Plass became applicable to large metagenomic datasets [Steinegger et al., 2019]. Besides, only a few tools that adopt the greedy assembly strategy for heterogeneous samples exist today. However, all of them apply targeted assembly. PRICE [Ruby et al., 2013] uses user-defined seeds as the starting point for the greedy approach. IVA [Hunt et al., 2015] picks seeds from the reads automatically and then compute a consensus assembly. The recently published assembly tool contigExtender [Deng and Delwart, 2021] employs the greedy strategy, starting with the contigs generated from metaSPAdes [Nurk et al., 2017].

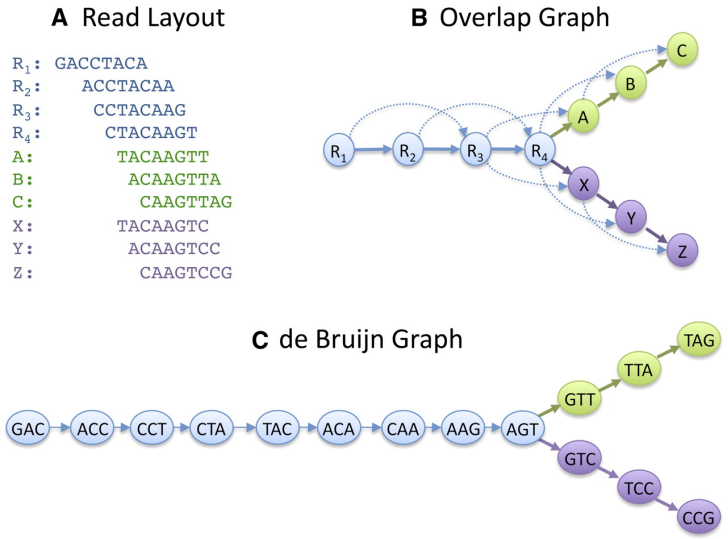


Figure 1.5.: Two graph-based assembly strategies. The Figure shows the data structures in the two main classes of assemblers. For a set of reads (A), the overlap graph (B) encodes reads as nodes and connects them if they have overlaps. Transitive edges are later removed to simplify the overlap graph towards a string graph. In the de Bruijn graph (C) each node represent a k-mer in the input reads. Nodes are connected if the represented k-mers occur consecutively in a read. Here, the k-mer size is set to 3. The Figure is taken from [Schatz et al., 2010].

1.4.2. Overlap-Layout-Consensus assembly

Another class of assembly algorithms is based on overlap graphs [Myers, 2005]. An overlap graph can be reconstructed from the set of sequencing reads. In the graph, each read is a node, and nodes are connected by an edge when the reads overlap (suffix prefix matches). See also figure 1.5 for a visualization. The (single genome) assembly problem is then equivalent to finding a path through the graph that contains each node exactly once (Hamiltonian path) [Pop, 2009]. Usually, overlap graph-based assemblers employ the Overlap-Layout-Consensus (OLC) strategy, which consists of three phases: overlap-phase, layout-phase, consensus-phase. In the first phase, the overlap graph is constructed by computing all pairwise read overlaps. This is time and memory consuming, as it scales quadratically in the naive implementation. In the layout-phase, OLC assemblers simplify the graph structure towards a string-graph [Myers, 2005]. Thereby, reads that are complete substrings of other reads ("contained reads") and transitive edges are removed. Contigs can then be obtained from the non-branching stretches of the graph. Finally, a majority voting is employed to obtain the consensus sequence for a contig (consensus-phase) via multiple sequence alignment of the reads that make up the contig. This allows to remove sequencing errors from the final contigs [Li et al., 2012].

OLC approaches were mostly used in the era of Sanger sequencing [Ayling et al., 2020]. The first assembler employing this approach was the Celera assembler [Myers et al., 2000]. However, OLC assemblers do not scale well with the higher throughput and shorter reads of today's Illumina sequencing platforms due to the need to compute and represent all pairwise overlaps [Pérez-Cobas et al., 2020]. The very high computational demand for large and complex metagenomic

datasets limits therefore the applicability of the OLC approach in metagenomic studies for short sequencing reads. With the current development of long read sequencing technologies, the OLC approach reemerges, but this is not described here further and can be found elsewhere [Rizzi et al., 2019].

Metagenomic assemblers that apply the OLC approach are for example Omega [Haider et al., 2014] and Genovo [Laserson et al., 2011]. Besides, specialized viral population assembler such as VICUNA [Yang et al., 2012] or SAVAGE [Baaijens et al., 2017] follow the OLC paradigm.

1.4.3. De Bruijn graph assembly

An alternative graph-based assembly approach is to use de Bruijn graphs (dBg), first introduced in the EULER assembler [Pevzner et al., 2001]. In this approach, the set of input reads is decomposed into substrings of length k, called k-mers, and a de Bruijn graph is utilized to store the relationship between the k-mers. The nodes represent the k-mers, and the edges between nodes indicate that two k-mers occur consecutively within the reads (see Fig. 1.5). Overlaps between the reads are therefore implicitly stored, and the (single genome) assembly task is defined as finding a path through the graph that contains each edge exactly once (Euler path) [Pevzner et al., 2001]. Finding an Euler path can be solved in linear time and is therefore more efficient than finding a Hamilton path in overlap graphs [Compeau et al., 2011b]. Furthermore, the linear scaling in edges enables the processing of large genomes [Compeau et al., 2011b] and even complex metagenomic data [Li et al., 2015]. Consequently, most of the recent work for metagenomic assemblers were done in this category, resulting in the development of several tools including MetaVelvet [Namiki et al., 2012], IDBA-UD [Peng et al., 2012], Ray-Meta [Boisvert et al., 2012], MEGAHIT [Li et al., 2015] and metaSPAdes [Nurk et al., 2017].

Constructing a de Bruijn graph starts kind of counterintuitive, as the reads are cut into k-mers and the read context gets lost. This is one major limitation of de Bruijn graph assemblers, as k-mers from a read must not necessarily lead to a linear path in the graph. Instead, shared k-mers can lead to complex graph structures. Especially in real metagenomic sequencing data, de Bruijn graphs can become quite complex including bubbles, tips and bulges resulting from repeats, closely related genomes and sequencing errors. This makes the graph traversing more challenging and requires various graph simplification heuristics [Ayling et al., 2020; Breitwieser et al., 2019]. Moreover, in contrast to the single genome assembler, where repeats and sequencing errors can be recognized due to coverage deviations, metagenomic assemblers have to deal with the highly uneven coverage across the different genomes.

In general, the most important parameter for the de Bruijn graph assembly paradigm is the choice of the size of k [Breitwieser et al., 2019; Chikhi and Medvedev, 2014]. A small k might not be specific enough for closely related genomes or short repeats and blow up the graph due to false edges. A large k, however, might miss connections in low coverage regions or due to the presence of sequencing errors and SNPs. This results in a sensitivity-specificity trade-off.

1.4.4. Limitations and challenges of assembly of viral genomes from metagenomic data

As already highlighted in the previous sections, assembling metagenomic data is quite challenging and all of the three approaches (greedy, OLC and dBg) have their own limitations, with dBg assemblers usually performing the best [Sczyrba et al., 2017; Vollmers et al., 2017]. Thereby, most of today's metagenomic assemblers were initially evaluated and optimized on bacterial metagenomic samples [Li et al., 2015; Namiki et al., 2012; Nurk et al., 2017; Peng et al., 2012; Sutton et al., 2019]. However, when considering the viral fraction of microbial communities, the assembly task becomes even more challenging. As described above, (1) viral genomes show high microdiversity and strain-level heterogeneity, resulting from high mutation rates [Minot et al., 2013; Warwick-Dugdale et al., 2019] and exchange of genomic regions through homologous recombination events [Simon-Loriere and Holmes, 2011] during co-infections [Cudini et al., 2019; Van der Kuyl and Cornelissen, 2007]. This enormous amount of diversity poses specific challenges. Moreover, (2) high background contamination from bacteria and eukaryotic sequences challenge the assembly of viral genomes further. Due to the small size of virus genomes, often only a small fraction of the reads from metagenomic samples originate from viral genomes, leading to low viral sequencing coverage [Moreno-Gallego et al., 2019; Roux et al., 2013; Shkoporov and Hill, 2019]. Also, (3) for viral enriched metagenomic data, the multiple displacement amplification (MDA) step can lead to extremes in the coverage biasing the data towards certain groups and insufficient coverage for low abundant genomes. Additionally, (4) viruses include many repeat regions within their genomes [Sutton et al., 2019]. Lastly, (5) viruses lack of a universal marker gene [Mirzaei and Maurice, 2017; Shkoporov and Hill, 2019], making subsequent identification of viral sequences from the assembly as well as assembly quality assessment more challenging. In total, the huge amount of sequencing data from large complex metagenomic samples together with large variability between viral genomes poses crucial challenges for current existing assembly methods.

As mentioned above, Greedy assemblers and OLC assembler have to compute all pairwise overlaps between the reads and/or contigs (greedy assemblers usually in each iteration, OLC assemblers to construct the overlap graph). As this typically scales quadratically in the number of reads, this is not applicable for large and complex metagenomic samples with millions to billions of reads due to time and memory issues [Vollmers et al., 2017]. Currently, methods that adopt a greedy/OLC approach are therefore mostly specialized tools to reconstruct viral genomes from single viral populations instead of whole metagenomes. This includes IVA [Hunt et al., 2015], VICUNA [Yang et al., 2012] and SAVAGE [Baaijens et al., 2017]. In contrast, de Bruijn graph assemblers like MEGAHIT [Li et al., 2015] and metaSPAdes [Nurk et al., 2017] can deal with the huge number of reads from metagenomic samples due to the k-mer splitting. They can store redundant read information more efficiently and are also much faster [Compeau et al., 2011a; Sutton et al., 2019]. However, they suffer from the above-mentioned specificity-sensitivity tradeoff, resulting in a loss of connectivity in the graph or complex graph structures. This creates the need for various graph simplification and filtering strategies to obtain contigs from the graph

structure. This results in short, fragmented contigs and loss of diversity (consensus assembly) [Sutton et al., 2019]. Aggressive collapsing as employed in the recently published viral metagenomic tools metaviralSPAdes [Antipov et al., 2020] and rnaviralSPAdes [Meleshko et al., 2022] can lead to longer contigs but cannot resolve the diversity.

In [Sutton et al., 2019], 16 metagenomic and viral specific assembly approaches were evaluated on simulated viromes, mock viral communities and a human gut virome. The authors point out that all assemblers suffered (to a varying degree) from the specific challenges of viral datasets resulting in low genome recovery, high degree of fragmentation and low quality contigs, especially in the case of low abundant genomes. Similar results have also been obtained in previous studies [García-López et al., 2015; Roux et al., 2017; Smits et al., 2014]. This creates the need for a new metagenomic assembler which can cope with viral diversity.

1.5. Sequencing error correction

Sequencing data analysis of NGS data usually starts with preprocessing the input read set to improve data quality. An initial step is sequencing error correction. Sequencing errors occur across all sequencing technologies, however the amount of sequencing errors as well as the dominating type varies significantly (see section 1.2). In general, there are three types of sequencing errors in NGS data: (1) substitution errors, where a nucleotide was erroneously replaced for another nucleotide, (2) insertion errors, where an additional nucleotide was erroneously introduced, and (3) deletion errors, where a nucleotide is missing. Furthermore, sequencers report 'N's if a nucleotide is unknown. Illumina sequencers basically avoid insertions and deletions, due to their one nucleotide per cycle approach, and substitution errors are clearly the dominating type [Schirmer et al., 2016]. Substitution errors can be caused by phasing effects, where individual clones desynchronize with the cluster and subsequently call the incorrect base [Pfeiffer et al., 2018]. Or they can be the result of cross-talk, either due to the overlap between the emission spectra of different nucleotides or from cross-talk between adjacent clusters [Pfeiffer et al., 2018]. Notably, phasing effects also increase the longer the sequencing run takes, which is also the reason why the number of errors increases toward the 3' end of the reads [Stoler and Nekrutenko, 2021] and why the sequence length is limited [Fuller et al., 2009].

Sequencing error correction is a fundamental step. Previous studies suggest that sequencing error correction can benefit assembly [Heydari et al., 2017], SNP detection [Kelley et al., 2010] and short read mapping [Qu et al., 2009]. To date, a number of error correction software exists and is reviewed in several publications, including [Alic et al., 2016; Heydari et al., 2017; Mitchell et al., 2020; Yang et al., 2013]. According to a popular benchmark study [Yang et al., 2013], sequencing error correction methods can be roughly divided into three categories: k-mer spectrum based, suffix tree/array based and multiple sequence alignment (MSA) based.

k-mer spectrum based. The simplest technique employs the distribution of sample-wise frequencies of k-mers appearing in the set of sequencing reads, which is termed the k-mer spectrum. It assumes a uniform sequencing coverage and relies on the idea that error-free k-mers occur roughly in the same amount as the sequencing coverage, whereas k-mers containing a sequencing error occur much more rarely (see Fig. 1.6). To decide if a k-mer is correct or not, the first step for these methods is to choose a frequency threshold. Such a threshold can be obtained by finding the local minimum of a fitted Poisson and Gaussian mixture model [Chaisson et al., 2009], by maximizing a likelihood function of these mixture models [Kelley et al., 2010], or by choosing the first minimum in the k-mer spectrum [Liu et al., 2013]. k-mers above this threshold are seen as correct and called "solid", k-mers below are likely to be erroneous and are called "weak". Sequencing errors are then corrected by turning weak k-mers into solid k-mers. Popular methods employing the k-mer spectrum approach include Musket [Liu et al., 2013], Quake [Kelley et al., 2010], Bloocoo [Benoit et al., 2014], Lighter [Song et al., 2014], BFC [Li, 2015], Reptile [Yang et al., 2010] and Bless [Heo et al., 2014]. The k-mer spectrum based approach is the most popular approach, probably due to its simplicity and scalability This is mostly resulting from the use of computationally efficient data structures, such as Bloom filters [Bloom, 1970] used for example in Lighter [Song et al., 2014], hash-tables or a combination of both [Li, 2015; Liu et al., 2013]. However, one of the problems with these methods comes when k-mers are underrepresented due to low coverage and are therefore erroneously filtered out. Especially on metagenomic datasets, the assumption of a uniform coverage is not fulfilled. Another problem arises from near-by errors resulting in k-mers that would require multiple changes within one k-mer. Further, most of the methods mentioned above apply greedy strategies and never revert their decisions.

Suffix tree/array based. Another technique is to employ suffix structures to hold the common parts of reads, and to identify sequencing errors as branches that have an unexpected low frequency. Methods using this technique generalize the k-mer based approach, as they can consider suffixes of variable length, instead of a fixed k-mer size. SHREC [Schröder et al., 2009] was the first method applying this approach by traversing a generalized suffix tree, whereas HiTEC [Ilie et al., 2011] utilizes the more memory efficient suffix array structure instead. Building on the same idea, Fiona [Schulz et al., 2014] utilizes a suffix array structure as well, but combines it with alignment computations to also consider insertion and deletion errors. However, in general, these methods are much more resource-consuming than the k-mer spectrum-based methods while often showing worse error correction performance [Yang et al., 2013].

MSA based. A third type of sequencing error correction methods relies on multiple sequence alignments (MSA). In a first stage, these methods group reads according to their location to the prospective reference, generate the MSA between the co-located reads and then correct reads according to the consensus obtained from the MSA. Two examples for this type of error correction tools are Coral [Salmela and Schröder, 2011] and ECHO [Kao et al., 2011]. Thereby,

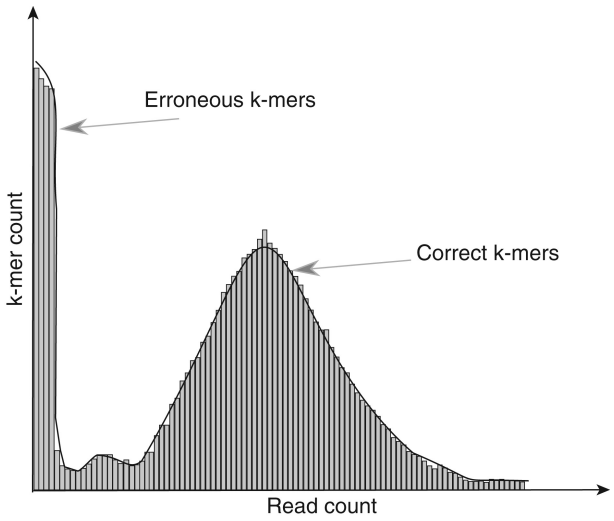


Figure 1.6.: Typical *k*-mers spectrum of sequencing reads obtained from a single genome. The distribution shows how often each k-mer frequency occurs in the reads. This distribution is utilized by the k-mer spectrum based methods to correct identify sequencing errors. They expect the erroneous k-mers to occur only in a small number of reads (first peak), whereas the frequency of true k-mers should occur close to the read coverage (bell shape). Notably, the peak of the bell shape is thereby shifted by $\frac{L-k+1}{L}$ with L being the size of the genome because the full k-mer must be covered by the read. Based on a threshold, set between the first peak and the bell-shape, allows them to distinguish the k-mers. Figure taken from [Alic et al., 2016].

Coral identifies read groups based on shared k-mers and aligns them using a variant of the Needleman–Wunsch algorithm [Needleman and Wunsch, 1970]. ECHO also utilizes shared k-mers for overlap detection and then uses a maximum a-posteriori estimation procedure to correct bases [Kao et al., 2011]. These methods have the advantage that they can consider the read as a whole instead of at the k-mer level or as suffixes of variable length. Moreover, they can make consistent corrections across overlapping reads. However, they are quite costly regarding run-time and memory-usage [Mitchell et al., 2020; Yang et al., 2013]. As they may overlap reads from different genomic locations, for example due to repeats, ambiguous correction can be performed.

Besides the mentioned categories, there are other techniques applied. For example, BayesHammer [Nikolenko et al., 2013] utilizes a hamming graph and Bayesian subclustering to identify erroneous k-mers and correct each nucleotide in a read based on the consensus of solid k-mers and cluster centers. Bcool [Limasset et al., 2020] constructs a de Bruijn graph, remove erroneous k-mers and subsequently maps sequencing reads to the cleaned de Bruijn graph to correct them. Further, some methods mentioned here can be included in more than one category. A comprehensive review of error correction methods for NGS data is given in [Alic et al., 2016]. Notably, almost all error correction tools were designed for single genome sequencing and their assumptions often do not hold on metagenomic samples.

1.6. Objectives and overview of this thesis

This thesis evolves around the development of two new software tools for the analysis of metagenomic sequencing data, with a focus on the viral fraction:

The first and main objective of this thesis is the development of a new metagenomic assembly method that can cope with the high viral diversity occurring in nature. With this method, we aim to assemble longer and more continuous fragments that provide more information about genes, gene complexes and ideally whole viral genomes than currently existing metagenomic assembly tools. Thereby, we aim to reveal more of the previously undiscovered viral diversity. This work is built on the protein level assembler Plass, which our group has previously published [Steinegger et al., 2019]. Its major contribution is the extension of Plass to a full nucleotide assembler that is guided by the amino acid sequences during the nucleotide assembly process. The resulting software tool is called PenguiN (protein guided nucleotide assembler) and is described in chapter 2.

The second objective of this thesis is the development of a new sequencing error correction method that can support the PenguiN assembler due to cleaner read overlaps. Precisely, with this method we aim to reduce the number of sequencing errors in the set of input reads while keeping the true (viral) diversity in the data. The method is implemented in a new tool called CoCo, which is described in chapter 3.

2. Development of a protein-guided nucleotide assembler and its application to viral metagenomic samples

In this chapter, I describe the new **p**rot**ein-gui**ded **n**ucleotide assembler, PenguiN, and present its application on viral metagenomic data. The main aim was to design a new de novo metagenomic assembler that can deal with the challenges of high viral diversity in complex metagenomic samples and therefore can assemble longer viral contigs than the current state-of-the-art metagenomic assemblers, ideally up to whole viral genomes. PenguiN is implemented in C++ and is part of the Plass software (https://github.com/soedinglab/plass). PenguiN is not yet published, but a manuscript is currently in preparation for the publication subsequent to this thesis. Thereby, parts of the following text and figures will be included.

2.1. Related work and underlying concept

Previously, our group developed the **p**rotein-**l**evel **ass**embler Plass [Steinegger et al., 2019], which assembles six-frame-translated sequencing reads into protein sequences. Assembling protein instead of nucleotide sequences has several advantages: (1) protein sequences are shorter, (2) most SNPs are synonymous or conservative, (3) repeats are shorter and less frequent, and (4) chimeric assemblies are less problematic as they do not lead to false conclusions about co-occurring genes [Steinegger et al., 2019].

With the aforementioned advantages, the enormous increase in the number of protein sequences that could be extracted from metagenomic samples was already shown in the original Plass publication [Steinegger et al., 2019]. Plass recovered 2-10 times more protein sequences from complex metagenomic samples than predicted from the nucleotide assemblies of the state-of-the-art metagenomic assemblers Megahit [Li et al., 2015] and metaSPAdes [Nurk et al., 2017]. Further, it generated non-redundant protein reference catalogs of 2 billion proteins from soil metagenomes (SRC) and 292 million proteins from marine eukaryotic metatranscriptomes (MERC). Since its publication, Plass has become highly accepted in the field. It was utilized for example in a metaproteomics study of soil samples from the Seine River (France) floodplain [Jouffret et al., 2021], used to assemble proteins from wild animal gut metagenomes [Youngblut et al., 2020], integrated in a pipeline to selectively assemble the neighborhoods extracted from large

metagenome assembly graphs [Brown et al., 2020] or used to recover protein sequences from samples of extreme natural environments such as geothermal hot springs and deep-sea ocean ecosystems in the context of the Virus-X project [Aevarsson et al., 2021]. Based on the advantages of protein-level assemblers, a recent study [Mirzaei et al., 2021] further emphasized the use of Plass also for viral metagenomic data, but also pointed out that the assembled proteins are not placed into a genomic context.

To be able to assemble long contigs up to whole viral genomes, we have to extend the contigs into the intergenic regions as well. During my doctoral research, I extended the protein-level metagenomic assembler Plass to bridge intergenic regions by guiding a nucleotide assembly using the protein-coding regions as seeds. Specifically, in a first stage, the underlying nucleotide sequences are assembled simultaneously to the protein sequences resulting in full open reading frames (ORFs), which can then be used as anchors and linked by aligning additional nucleotide reads in a second step. See also Fig. 2.1a for a graphical explanation of this approach. This protein-guided nucleotide assembly approach is now implemented in the new software tool PenguiN, which I integrated into the Plass software. We believe that this approach has the advantage to simplify the assembly problem, as genomes, which already differ on amino acid level do not have to be compared on the longer and therefore more computational expensive nucleotide sequences. At the same time, single nucleotide mutations in the coding regions, which do not change the amino acid sequence significantly (synonymous or conservative mutations), do not hinder the assembly process. While, genomes that have picked up multiple changes (lead to different strains), can be distinguished due to co-occuring mutations within full-read overlaps.

2.2. Algorithm and Implementation

2.2.1. Outline of the PenguiN algorithm and key ideas

PenguiN proceeds in two main stages, which I briefly outline here. The details are then described in section 2.2.2 and the whole workflow of the software tool is depicted in Fig. 2.1. The guided assembly (stage I) assembles six-frame translated reads to proteins and co-assembles the corresponding nucleotide ORFs. The nucleotide assembly (stage II) links the assembled ORFs using the original nucleotide reads. In both stages, PenguiN uses thereby the same graph-free, greedy iterative assembly strategy as developed for Plass [Steinegger et al., 2019].

2.2.1.1. Assemble six-frame translated reads to proteins and co-assemble nucleotide ORFs

First, PenguiN assembles the six-frame translated reads analogous to the original Plass algorithm by (1) finding overlaps in linear times, and (2) iteratively extending the sequences on both sides using full-alignment overlaps. Afterwards (3), extended and unchanged sequences are used as input for the next iteration and the steps are repeated.

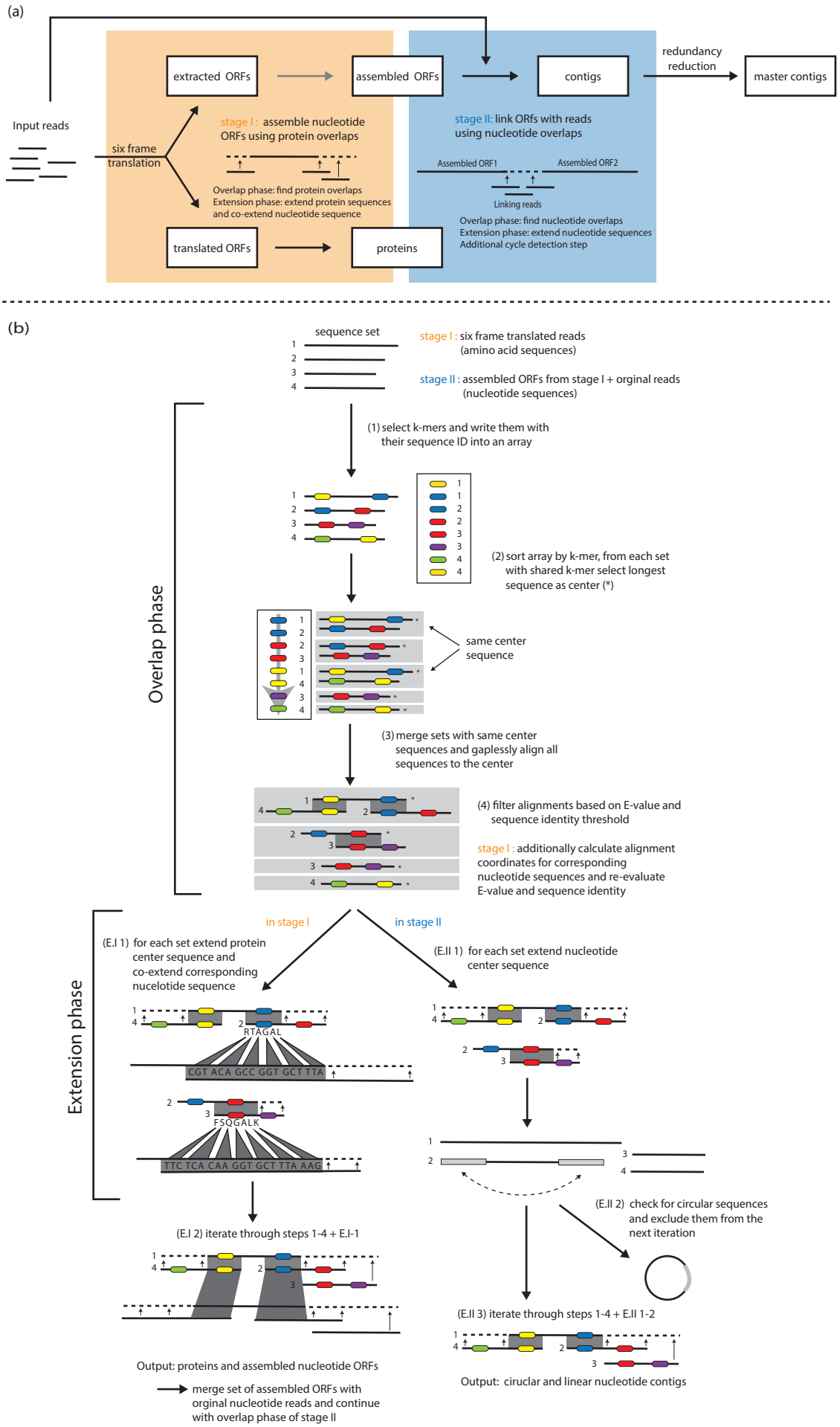


Figure 2.1.: Caption on the next page

Figure 2.1.: Overview of the workflow and algorithm of PenguiN. **(a)** PenguiN proceeds in two main stages. The guided assembly (stage I) assembles six-frame translated reads into proteins and thereby co-assembles the underlying nucleotide sequences, resulting in complete open reading frames (ORFs). Assembled ORFs from stage I are then linked with the original nucleotide reads within the nucleotide assembly (stage II), resulting in long continuous stretches (contigs). At the end, a redundancy reduction step is performed, clustering redundant contigs or contigs with minor differences to the final master contigs. **(b)** Both stages thereby consist of two phases: overlap phase and extension phase, supplemented by an additional cycle detection step within stage II. This panel summarizes the workflow and highlights the differences in each step between the stages. A more detailed description of the steps is given in the main text (section 2.2.2).

The key idea of the PenguiN assembly algorithm is now, that during the whole process the underlying nucleotide sequence can be assembled simultaneously. Whereas the overlaps are found and evaluated in linear time on the protein sequences, they are re-evaluated on the nucleotide sequence as well and only processed further, if they fulfil an E-value and a minimum sequence identity threshold as well for the corresponding nucleotide alignment (overlap phase). During the subsequent extension phase of the protein sequences, the corresponding nucleotide sequences are then extended in parallel. Therefore, stage I is in the following referred to as the guided assembly part.

To avoid code duplication, PenguiN modifies approaches implemented in Plass. The details are described in section 2.2.2. After 5 (default) iterations through overlap and extension phase, the process is terminated, and the extended sequences should reflect the protein and nucleotide sequences of full open reading frames. In contrast to the Plass algorithm, proteins translated in the wrong frame are thereby not an issue, as for the subsequent steps only the nucleotide sequence will be processed further.

2.2.1.2. Link ORFs with reads

The second stage then aims to link the assembled ORFs from the first stage using the original reads to bridge non-coding regions. Thereby, PenguiN uses the nucleotide sequences of the assembled ORFs from the first stage together with the original reads as input and performs a nucleotide (only) assembly mostly following the same general scheme as used in stage I for the protein and ORF assembly: (1) First it looks for overlaps in linear time, but this time on nucleotide level, and (2) then iteratively extends the nucleotide sequences using full-alignment overlaps (see Fig. 2.1). In the following, this part is referred to as the iterative nucleotide assembly stage. Also here, parts of the original Plass code were re-used but needed to be adapted or extended to nucleotide sequences. The details can be found in section 2.2.2.

However, since we also aim to assemble much longer contigs up to whole viral genomes in stage II, the issue of circular genomes and long terminal repeats at the ends of linear viral genomes of the form RAR arose. Both are common genome structures in viruses. Of 8420 sequences from viruses in the RefSeq [O'Leary et al., 2016] database (viral.1.1.genomic.fna from 08/2019), 2119 were labelled as "circular" in the corresponding GenBank annotation file (viral.1.genomic.gbff)

and further 350 showed a terminal repeat of length ≥ 22, exceeding PenguiN's default k-mer size of 22.

In contrast to graph-based assemblers, which can detect these structures as cyclic paths [Compeau et al., 2011b], our graph-free assembly strategy would not recognize if a contig started to repeat itself. Our greedy iterative overlap-based assembly strategy would just overextend the contigs that present such structures in the following iterations. To overcome this issue in PenguiN, I implemented an additional step to detect circular structures after each extension phase in stage II using a heuristic similarity approach (see section 2.2.2.5). Contigs containing circular structures are thereby marked as complete and filtered out before starting the next iteration.

All other non-complete sequences, extended and unchanged, are then used as input for the next iteration within the nucleotide assembly stage. By default, the whole process is also repeated 5 times. Finally, all circular complete sequences collected over all iterations as well as all (circular and linear) sequences resulting from the last iteration are pooled together and passed to a final redundancy reduction step, if they have been extended at least once and fulfilling a minimum length threshold (default: 500 bp). The redundancy is then reduced using an adapted version of the Linclust [Steinegger and Söding, 2018] algorithm (see section 2.2.2.6) and cluster representatives are output as final master contigs, each representing a group of very similar sequences, which only differ in sequencing errors or in a minor number of mutations.

2.2.2. Algorithm and software details

As described, both main assembly stages, the guided assembly (stage I) and the nucleotide assembly (stage II), consist of an overlap phase and an extension phase, supplemented by a cycle detection procedure in the nucleotide assembly, which is iteratively repeated. Before the actual assembly process starts, paired-end reads are merged and six-frame translation is performed. After the assembly process is completed, the redundancy is reduced. In the following, I now explain the details of all these steps and describe some optimizations to the core algorithm.

2.2.2.1. From input reads to potential ORFs

Input format and merging reads

PenguiN expects short sequencing reads as input, either paired-end reads in FASTQ format or single reads in FASTQ or FASTA format. However, due to the large size of typical metagenomic samples, sequencing reads (and later contigs) are not stored in memory. Instead, they are transformed into the efficient MMseqs2 [Steinegger and Söding, 2017] database format that makes direct byte access to specific sequences possible, while also avoiding random file system accesses as good as possible (see section A.3 for a description of the database format). Further, it is crucial for efficient parallelization of the software as it allows for thread-safe read-write access in the next steps. Analogous to Plass, PenguiN converts the reads directly into the MM-

seqs2 database format in case of single-end reads or, in case of paired-end reads, first merges overlapping paired-end reads into longer sequences utilizing the code of FLASH [Magoč and Salzberg, 2011]. Subsequently, all merged and non-merged reads are output together as a MMseqs2 database. With the latter, we make use of the pairing information to get already a bit longer sequences and therefore also longer and more significant overlaps in the subsequent assembly process.

Extract and translate ORFs

In the next step, analogous to Plass, PenguiN extracts all six open reading frames (ORFs) with at least 45 codons from the reads as well as all ORFs with at least 20 codons starting with a putative ATG start codon, i.e. the first ATG codon after a stop codon in the same frame. Afterwards, extracted ORFs are translated into the corresponding protein sequences using the canonical genetic code table. However, alternative codon tables can be specified. During the translation process, stop codons are marked by an asterisk (*) and putative ATG start codons are prepended with an asterisk before the methionine residue.

2.2.2.2. Finding Overlaps in linear time

Finding overlaps in linear time is a crucial step in the PenguiN assembly algorithm. In stage I PenguiN searches for overlaps between the six-frame translated sequences on amino acid level and later transform them to the corresponding nucleotide sequence. In stage II, it searches for overlaps on nucleotide level directly. However, a pairwise comparison of all sequences would lead to quadratic runtime complexity. This would not be feasible for a typical metagenomic dataset containing 100 million of reads. To overcome this in Plass, the linear time clustering algorithm Linclust [Steinegger and Söding, 2018] was adapted. PenguiN uses the same strategy with small modifications on the protein-level and adaptions and extensions on the nucleotide-level. In the following, I describe the procedure in more detail (Fig. 2.1b, steps 1-4), and point out the most important adaptions. However, the reader is also referred to the original Linclust [Steinegger and Söding, 2018] and Plass publication [Steinegger et al., 2019].

First, k-mers are extracted from each sequence (Fig. 2.1b, step 1). But in contrast to Plass, the number of extracted k-mers per sequence is not fixed (default: 60) but scales linearly with the sequence length ($m + \lambda \times n$), with n being the sequence length, λ the scaling factor and m a constant (default: $m = 60$, $\lambda = 0.1$). This was introduced to compromise for the huge length differences between the sequences occurring during the assembly iterations, especially in stage II where sequences ranges from the length of a read ($\sim$150 bp) to full (viral) genome lengths ($\sim$ 2–100 kbp). The selection of the k-mers is thereby based on the lowest values of a hash function to avoid positional clustering of the selected k-mers and picking different k-mers from each sequence. By choosing the XXH64 hash function from https://github.com/Cyan4973/xxHash that maps similar k-mers to uncorrelated hash values, a quasi random distribution of the extracted k-mers

across a sequence can be achieved. At the same time, selecting the k-mers with the lowest hash value ensures that the same k-mers tend to be selected from similar sequences.

Within stage I, k is set to 14 analogous to Plass and the same reduced amino acid alphabet with $|\mathcal{A}| = 13$ letters representing the following groups of amino acids (L, M), (I, V), (K, R), (E, Q), (A, S, T), (N, D) and (F, Y) is used. In stage II, k is set to 22 and nucleotide sequences are represented with the full nucleotide alphabet $\mathcal{A} = \{A, C, G, T\}$. However, due to the unknown orientation of the nucleotide sequences, also the reverse complement of each k-mer has to be considered and the k-mer index is chosen as the canonical representation (the lower index of the forward and reverse complement k-mers). k-mers containing a wildcard symbol, represented as 'X' on amino acid level or 'N' on nucleotide-level, palindromic k-mers as well as k-mers occurring multiple times in a sequence are skipped as they would lead to ambiguous matches in the subsequent steps.

For each selected k-mer, the k-mer index (8 bytes) is saved in an array along with the sequence identifier (4 bytes), the length of the sequence (2 bytes) it was extracted from and the position in that sequence (2 bytes). This allows for a memory efficient representation. However, to ensure the correct representation for long sequences in the later nucleotide iterations as well, I introduced a template type. If the longest sequence in the current iteration is larger than what can be represented with 2 bytes (65 535), the last two values are expanded to 4 bytes dynamically.

After selecting and storing the k-mers with the lowest hash values, they are sorted by the k-mer index and sequence length to find sequence sets containing the same k-mer (Fig. 2.1b, step 2). For each set, the longest sequence is picked as the center sequence. For all other sequences sharing this k-mer (member sequences), the matching diagonal is calculated as $i - j$, where i is the k-mer position in the center sequence and j the k-mer position in the member sequence. Then, for each member sequence, information about center sequence identifier and matching diagonal are stored, replacing the k-mer index and position fields in the array. This allows to find and merge sets with the same center sequence and to remove duplicate pairs of center and member sequences when sorting the array now by the center sequence identifiers. If multiple matches between the same center and member sequence are present, only the diagonal with the higher number of k-mer matches is kept.

Afterwards, for each group, k-mer matches are extended to the full ungapped alignment between the center sequence and each member sequence along the stored diagonal (Fig. 2.1b, step 3). Originally, this was done in Plass using one-dimensional dynamic programming to find a local alignment on the matching diagonal. PenguiN, and also the newest Plass version, now rescores the full diagonal, allowing for a more robust evaluation of the overlapping region. However, if a sequence contains an asterisk in stage I (representing the beginning or end of an ORF) the alignment is truncated to not contain this character. This ensures that the assembly on protein-level will later stop at these boundaries. Then, for each alignment, the sequence identity value is computed and an E-value is estimated using the ALP library [Sheetlin et al., 2016] and, by

default, the Blosum62 substitutions matrix on protein-level and a match score 2 and mismatch score -3 on nucleotide-level.

Alignments satisfying an E-value (default: 10^{-5}) and a sequence identity cutoff (default: 97% at protein-level, 99% at nucleotide level) are then used in the subsequent extension phase. The alignments represent the overlaps, while the hang-offs of the member sequences represent the possible extensions for the center sequence.

2.2.2.3. Transform protein alignments to corresponding nucleotide alignments

To make the co-assembly on nucleotide level within stage I possible, PenguiN performs an additional step to transform each protein alignment between center and member sequences into the corresponding nucleotide alignment. Thereby, I do not re-calculate the k-mer matches, but only transform the alignment coordinates and re-calculate the sequence identity value by counting the matches on nucleotide level and divide them by the alignment length. In this way, the k-mer matches on protein level operate as seeds for the alignments on nucleotide level. Furthermore, the E-value of the nucleotide alignment is recalculated using again the ALP library [Sheetlin et al., 2016] and, by default, a match score of 2 and a mismatch score of -3. Only alignments that still satisfy the E-value and the nucleotide sequence identity cutoff (default: 99%) are then considered as overlaps between center-member pairs in the subsequent extension stage.

2.2.2.4. Greedy extension strategy based on a Bayesian model

In the extension step, each center sequence is extended by concatenating the non-overlapping residues of the member sequences on each side ("hang-offs"), either on protein and nucleotide level simultaneously (stage I), or on nucleotide level only (stage II). See also Fig. 2.1b.

Even though only the member sequences with alignments to the center sequence satisfying the sequence identity threshold and the E-value criterion are thereby considered, multiple extensions might be possible on each side of a center sequence. In the Plass algorithm, the list of alignments is sorted in order of descending overlap sequence identity and the best left and right extensions are chosen. However, during my doctoral research I realized that this strategy is not ideal when the length of the compared overlaps vary greatly, which is particularly seen during the nucleotide assembly or in later iterations of the guided assembly.

For example, an overlap with length 1000 and sequence identity 98.9% might be more trustworthy than an overlap of length 100 and sequence identity 99.0% as a longer overlap also yields a statistically more significant estimation of the actual similarity between the full sequences. In PenguiN, I resolved this issue with a new alignment sorting strategy using a Bayesian formulation of the problem, which can be solved analytically. For the underlying mathematical formulation of the problem, I was supported by Johannes Söding and Étienne Morice.

Given a query sequence (center sequence) and a target sequence (member sequence) with a mean fraction of non-identical residues q_c, and an alignment of length M_c between them, out of which m_c are mismatches and $M_c - m_c$ are matches, the probability distribution for the number of mismatches in the alignment is a binomial distribution

$$p(m_c|q_c, M_c) = \text{Binom}(m_c|q_c, M_c) = \binom{M_c}{m_c} q_c^{m_c}(1-q_c)^{M_c-m_c}\,. \tag{2.1}$$

The q_c are hidden variables while M_c and m_c are the observed variables. Given the latter, the probability distribution of the former can be deduced using *Bayes' theorem*,

$$p(q_c|m_c, M_c) = \frac{p(m_c|q_c, M_c)\,p(q_c)}{\int p(m_c|q, M_c)\,p(q)\,\mathrm{d}q}\,, \tag{2.2}$$

where $p(q)$ is the prior probability for the fraction of mismatches of the alignment. We can model $p(q)$ as a beta distribution, defined by

$$p(q) = \text{Beta}(q|a_q, b_q) = B(a_q, b_q)^{-1}\, q^{a_q-1}(1-q)^{b_q-1} \tag{2.3}$$

with

$$B(a_q, b_q) = \frac{\Gamma(a_q)\Gamma(b_q)}{\Gamma(a_q+b_q)}\,. \tag{2.4}$$

We therefore obtain for the hidden q_c

$$\begin{aligned} p(q_c|m_c, M_c) &\propto p(m_c|q_c, M_c)\,p(q_c) \\ &= \binom{M_c}{m_c}\frac{\Gamma(a_q+b_q)}{\Gamma(a_q)\Gamma(b_q)}\, {q_c}^{m_c+a_q-1}(1-q_c)^{M_c-m_c+b_q-1} \\ &\propto \text{Beta}(q_c|m_c+a_q, M_c-m_c+b_q)\,. \end{aligned} \tag{2.5}$$

The proportionality constant between the left and right-hand sides must be 1 since both are normalized probability distributions, and therefore

$$p(q_c|m_c, M_c) = \text{Beta}(q_c|m_c+a_q, M_c-m_c+b_q)\,. \tag{2.6}$$

Given now two possible extensions for the same contig, described by the alignments (m, M) and (m', M'), the probability that the first target sequence has lower dissimilarity to the query contig than the second, is then given by

$$p(q < q'|m, M, m', M') = \int_0^1 \int_q^1 p(q|m, M)\, p(q'|m', M')\, \mathrm{d}q'\, \mathrm{d}q \tag{2.7}$$
$$= \int_0^1 \int_q^1 \mathrm{Beta}(q|m+a_q, M-m+b_q)\, \mathrm{Beta}(q'|m'+a_q, M'-m'+b_q)\, \mathrm{d}q'\, \mathrm{d}q$$

using formula (2.6).

We therefore need to compute the integral

$$I := \int_0^1 \int_q^1 q^{\alpha-1}(1-q)^{\beta-1} q'^{\alpha'-1}(1-q')^{\beta'-1}\, \mathrm{d}q'\, \mathrm{d}q \tag{2.8}$$

for

$$\begin{aligned} \alpha &:= m + a_q, \\ \beta &:= M - m + b_q, \\ \alpha' &:= m' + a_q, \\ \beta' &:= M' - m' + b_q. \end{aligned} \tag{2.9}$$

By substituting $q' = t + (1-t)q$; $t \in [0,1]$ with $\mathrm{d}q' = (1-q)\,\mathrm{d}t$, the integral can be rewritten with fixed boundaries,

$$I = \int_0^1 \int_0^1 q^{\alpha-1}(1-q)^{\beta-1}(t+(1-t)q)^{\alpha'-1}\, [(1-q)(1-t)]^{\beta'-1}\, (1-q)\, \mathrm{d}t\, \mathrm{d}q\,. \tag{2.10}$$

Expanding

$$(t+(1-t)q)^{\alpha'-1} = \sum_{i=0}^{\alpha'-1} \binom{\alpha'-1}{i} (1-t)^i\, q^i\, t^{\alpha'-1-i} \tag{2.11}$$

yields

$$\begin{aligned} I &= \sum_{i=0}^{\alpha'-1} \binom{\alpha'-1}{i} \int_0^1 \int_0^1 q^{\alpha+i-1}(1-q)^{\beta+\beta'-1}\, t^{\alpha'-i-1}(1-t)^{\beta'+i-1}\, \mathrm{d}t\, \mathrm{d}q \\ &= \sum_{i=0}^{\alpha'-1} \binom{\alpha'-1}{i} \mathrm{B}(\alpha+i, \beta+\beta')\, \mathrm{B}(\alpha'-i, \beta'+i)\,. \end{aligned} \tag{2.12}$$

Using $n! = \Gamma(n+1)$, the probability

$$p(q < q'|m, M, m', M') = \mathrm{B}(\alpha, \beta)^{-1} \mathrm{B}(\alpha', \beta')^{-1}\, I \tag{2.13}$$

can then be written as

$$p(q < q'|\ldots) = \sum_{i=0}^{\alpha'-1} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\alpha'+\beta')}{\Gamma(\alpha')\Gamma(\beta')} \frac{\Gamma(\alpha')}{\Gamma(i+1)\Gamma(\alpha'-i)} \frac{\Gamma(\alpha+i)\Gamma(\beta+\beta')}{\Gamma(\alpha+\beta+\beta'+i)} \frac{\Gamma(\alpha'-i)\Gamma(\beta'+i)}{\Gamma(\alpha'+\beta')}$$
$$= \sum_{i=0}^{\alpha'-1} \frac{\Gamma(\alpha+\beta)\Gamma(\alpha+i)\Gamma(\beta+\beta')\Gamma(\beta'+i)}{\Gamma(\alpha)\Gamma(\beta)\Gamma(i+1)\Gamma(\alpha+\beta+\beta'+i)\Gamma(\beta')} . \tag{2.14}$$

With a constant C and a term π_i depending on i,

$$C = \frac{\Gamma(\alpha+\beta)\Gamma(\beta+\beta')}{\Gamma(\beta)\Gamma(\alpha+\beta+\beta')} \tag{2.15}$$

$$\pi_i = \frac{\Gamma(\alpha+i)\Gamma(\beta'+i)\Gamma(\alpha+\beta+\beta')}{\Gamma(i+1)\Gamma(\alpha)\Gamma(\beta')\Gamma(\alpha+\beta+\beta'+i)} , \tag{2.16}$$

the probability $p(q < q'|m, M, m', M') = \sum_{i=0}^{\alpha'-1} C\,\pi_i$ can then be computed iteratively by

$$\pi_0 = 1 \tag{2.17}$$

$$\pi_i = \pi_{i-1} \times \frac{(\alpha+i-1)(\beta'+i-1)}{i(\alpha+\beta+\beta'+i-1)} = \pi_{i-1} \times r_i . \tag{2.18}$$

All together, the probability $p(q < q'|m, M, m', M') = \sum_{i=0}^{\alpha'-1} C\,\pi_i$ can then be computed using the following pseudocode.

```
1 Function Prob(α, β, α′, β′):
2     C = (Γ(α + β)Γ(β + β′))/(Γ(β)Γ(α + β + β′));
3     sum = pi = 1.0;
4     for i = 1.0 ... α′ − 1 do
5         pi *= (α+i-1) * (β′+i-1) / i / (α+β+β′+i-1) ;
6         sum += pi
7     end
8     return C * sum;
```

Algorithm 1: Naive implementation

However, to ensure numerical stability, the terms are calculated in log-space in the current PenguiN implementation:

```
1 Function Prob_log(α, β, α′, β′):
2     lnC = ln Γ(α + β) + ln Γ(β + β′) − ln Γ(β) − ln Γ(α + β + β′);
3     lnpi = 0.0;
4     sum = exp(lnC);
5     for i = 1.0 ... α′ − 1 do
6         lnpi += ln(α+i-1) + ln(β′+i-1) - ln(i) - ln(α+β+β′+i-1) ;
7         sum += exp(lnpi +lnC)
8     end
9     return sum;
```

Algorithm 2: Numerically stable implementation

Based on the resulting probability value, the following decisions are made:

If $p(q < q'|m, M, m', M') > 0.55$ the (m, M) alignment is preferred, if $p(q < q'|m, M, m', M') < 0.45$ the (m', M') alignment is preferred. If the probability falls into the range 0.45 to 0.55 we consider it as "inconclusive", and prefer the extension that offers the longer extension instead.

Using this formulation to compare two alignments, PenguiN sorts for each center sequence the whole list of alignments of its member sequences. Then, the list of alignments is processed in the resulting order and the best left and right extension is chosen (Fig. 2.1b, extension phase). Afterwards, the extended center sequence is realigned to the not yet processed member sequences, which show hang-offs still exceeding the extended center sequence on one of the sides. The alignment list is iterated further until no more extension can be made.

2.2.2.5. Identify circular contigs

As described above, circular and long terminal redundant genome structures are common among viruses. This made the introduction of a cycle detection step (Fig. 2.1b, step E-II.2) in PenguiN necessary to avoid over-extension of contigs representing such structures. Thereby, I do not distinguish between circular and terminal redundant genomes, as, according to the short read sequencing, reads from both behave similarly.

The first naive way I implemented to identify a contig that is starting to repeat itself was to align the sequence with itself. However, calculating such an alignment has a quadratic runtime complexity with the length of the sequence and is therefore computational demanding when applied for each sequence after each iteration. On the other hand, performing the cycle detection only at the very end would mean that circular sequences would be extended further and further within the iterative assembly procedure and therefore computing time and memory to extract k-mers, calculating alignments and extending these sequences would be spent unnecessarily in each iteration, slowing down the whole assembly process. Thus, I introduced a heuristic, efficient procedure in PenguiN to identify circular sequences after each iteration that (1) limits the region to align and (2) approximates the alignments with k-mer matches.

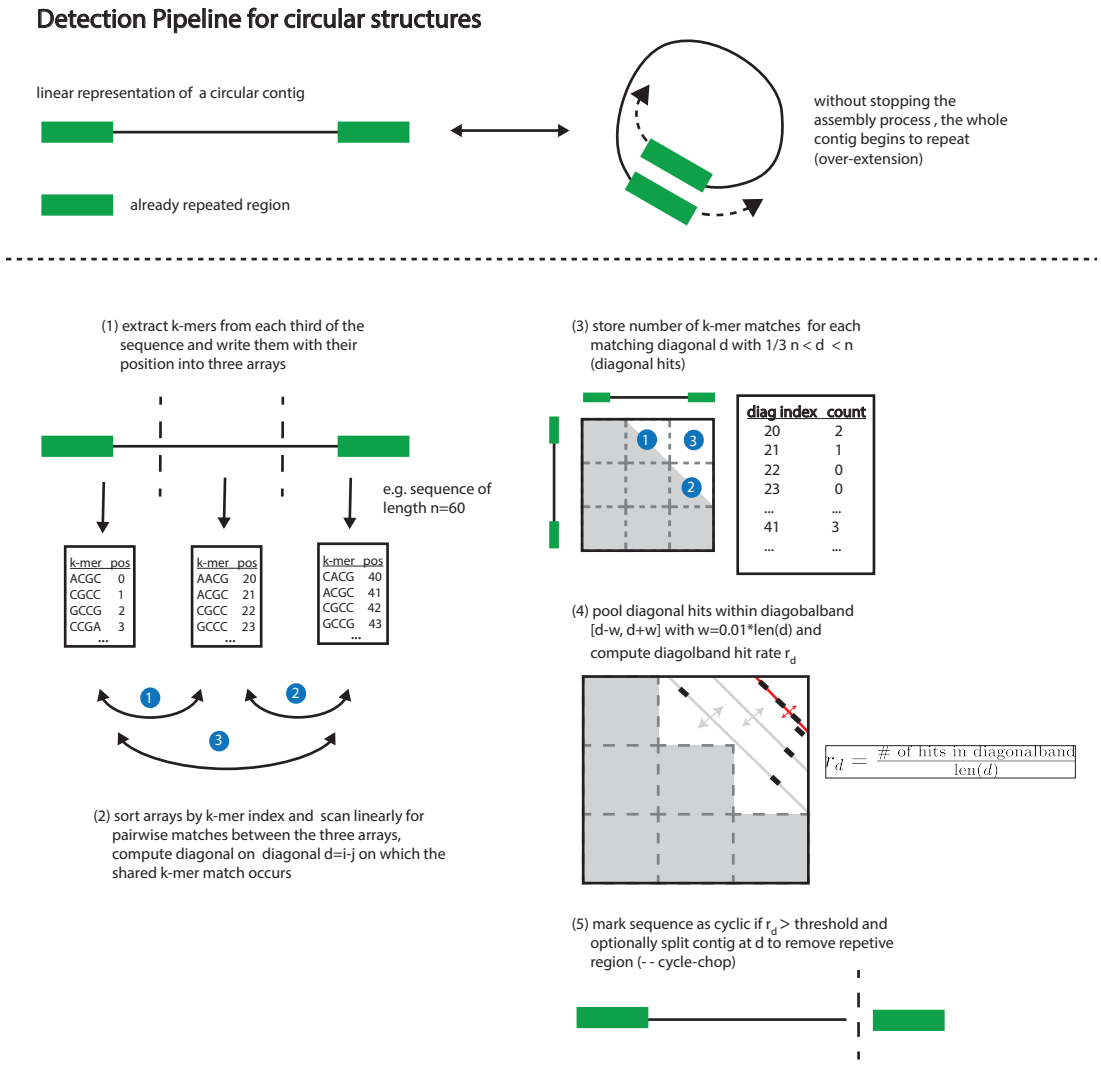


Figure 2.2.: Detection pipeline for circular sequences. To avoid overextension of contigs representing circular structures during the iterative assembly, I introduced a heuristic procedure in PenguiN to identify and exclude them from subsequent assembly iterations. First, k-mers from each third of the sequence are extracted, and stored in separate arrays as a k-mer index together with its position in the sequence. The three arrays are sorted by the k-mer index and compared pairwise to identify shared k-mers between the three sets. For each diagonal $d > 1/3n$, where n is the length of the sequence, the number of shared k-mers is then counted as diagonal hits. Then, for each diagonal d the hits from all diagonals within a distance of 1% of the length of the diagonal are pooled, getting the hits for diagonal bands. For each of these diagonal bands, a hit rate r_d is calculated as the fraction of number of hits within the diagonal band and the length of the diagonal d. If the hit rate r_d overcomes a certain threshold (default: 0.24), the contig is considered as circular and excluded from subsequent assembly iterations. Optionally, the repetitive region of a circular contig is removed by splitting the contig at position d', where d' is the longest diagonal fulfilling $r_d > 0.24$.

The limitation to only align parts of the sequences was possible due to the design of the extension phase. In each iteration, a sequence can be extended on both sides. However, the sequence to extend (center sequences) is always longer than or equal in size to the member sequences that contribute their non-overlapping residues for the extension. Therefore, a sequence with non-identical ends cannot contain more than three copies after one more iteration. Considering the alignment task as a dynamic programming problem [Needleman and Wunsch, 1970], as

is common, this means a reduction of the computation to one third of the distance matrix (see Fig. 2.2). To reduce computation time further, I designed a procedure to approximate the alignments between the thirds of the sequences by counting k-mer matches instead of computing the full alignments. More precisely, the number of cumulative k-mer matches between the thirds of the sequence within diagonal bands is determined and the respective contig is marked as circular complete, if this number is significant compared to the length of the diagonal band (hit rate r_d). The whole detection workflow is depicted in Fig. 2.2. Using 5994 viruses from the RefSeq [O'Leary et al., 2016] database, I found a hit rate threshold of 0.24 working well to distinguish terminal redundant/circular and linear sequences (see Fig. A.1 in the appendix).

Circular contigs do not participate in the subsequent iterations, instead they are collected over all iterations and written to the final output file. To mark where circular completeness for the user, they get a cycle flag in their header in the final output file.

2.2.2.6. Redundancy reduction

The last step in PenguiN, before writing the final assembly file, is the redundancy reduction step. PenguiN reduces the redundancy within the set of assembled contigs using the linear-time clustering algorithm Linclust [Steinegger and Söding, 2018] with cluster mode 2 (greedy incremental), which is the analogous mode to the CD-HIT clustering algorithm [Fu et al., 2012], and parameters set to a minimum sequence identity of 97% at a minimum 99% coverage of the shorter sequence (options --cluster-mode 2 --cov-mode 1 -c 0.99 --min-seq-id 0.97). Each cluster then represents a set of redundant or very similar sequences diverging $< 3\%$, from which only the longest sequence (cluster representative) is output as the master contig.

Linclust was originally implemented to cluster protein sequences, but was later extended to nucleotide sequences as well [Steinegger and Söding, 2018]. First, it identifies possible sets of similar sequences based on shared k-mers. Afterwards, within each set, it compares each sequence only with the longest sequence (center sequence) and clusters them in three consecutive clustering steps using the Hamming distance, the ungapped local alignment and the gapped local alignment.

During my doctoral research I contributed to the improvement of Linclust for nucleotide sequences by fixing smaller bugs in the software especially occurring for long nucleotide sequences and adding a template type for the storage of sequence length and sequence position making it applicable for our long assembled contigs. Further, I expanded Linclust to the use-case in PenguiN by adding a logic to cluster contigs from circular genomes (set with the parameter --wrapped-scoring).

Previously, two sequences representing the same circular genome but with different starting points would not cluster together. Let contig A and B be two sequences representing the same circular genome, but with different starting points. They would not align continually, but instead showing two alignments, one from the beginning of contig A to the end of contig B, and one

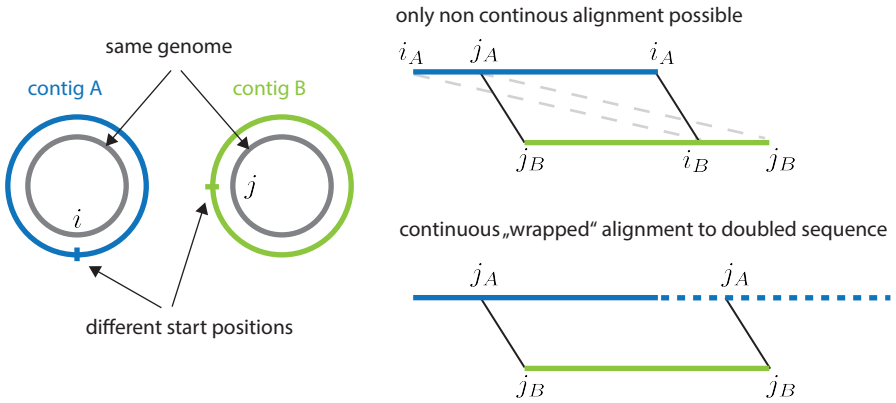


Figure 2.3.: Alignment computation for contigs originating from circular or terminal redundant genomes. The same genome can be described with contigs starting at different positions within the genome. The pairwise alignment between such contigs would be non-continuous (see top right). Therefore, I adapted the alignment computation step during the redundancy reduction by doubling the query sequence. This allows for a continuous alignment between the two contigs (see bottom right).

from the beginning of contig B to the end of contig A (see Fig. 2.3). Usually, neither of the two alignments fulfills the coverage threshold alone. However, due to the data structures and algorithms implemented in Linclust it was not possible to obtain multiple alignments, which could be checked for this special kind of "wrapped" continuity. When using the wrapped-scoring option now, one of the sequences (query sequence) is doubled before the actual alignment calculation, resulting in a continued alignment beyond the original ends of the query sequence (see Fig. 2.3). This alignment allows now for correct sequence identity and E-value calculation between contig A and B within the Linclust algorithm, whereas coverage is still computed with the original query sequence length.

The adapted version of Linclust was integrated into the PenguiN pipeline as it is magnitudes faster than clustering methods that perform all-against-all comparisons. However, Linclust is less sensitive than the latter [Steinegger and Söding, 2018]. Therefore, it might be necessary to perform an additional, more sensitive clustering step, depending on the use case. One option is to run the new MMseqs2 nucleotide_clustering module.

2.2.2.7. Handling sequencing errors

PenguiN does not come with its own error correction step. Therefore, I allow for a small number of mismatches within the overlap regions. On the protein-level, the sequence identity threshold is set to 97% (default) and on the nucleotide-level it is set to 99% (default), each accounting for approximately 0.5% sequencing errors within the aligned sequences. Further, due to the final redundancy reduction step, contigs diverging $\leq 3\%$ are clustered together and represented by the same master contig. Therefore, contigs that only differ due to sequencing errors are most likely to end up in the same cluster. The values can also be changed by the user to take higher error rates into account. However, this would also increase the chance for misassembled (chimeric)

contigs and is therefore not recommended. Further, PenguiN requires at least the length of a k-mer to match in order to identify an overlap. The more errors there are in the reads, the less likely there will be overlaps between the reads.

Usually, PenguiN will work on uncorrected datasets. However, cleaner data might improve the assembly, especially as Illumina technology mainly introduces errors at the ends of the reads leading to cumulative sequencing errors in the overlap regions [Bolger et al., 2014; Stoler and Nekrutenko, 2021], which might result in fragmented assemblies in case of small coverage values or low abundant strains. During my doctoral research, I developed CoCo, which I describe in chapter 3, to correct the data before applying PenguiN. However, any sequencing error correction tool can be used in principle if it is accurate enough.

2.2.2.8. Software availability and documentation

PenguiN is implemented in C++ and has been integrated into the free GPLv3-licensed Plass software. It makes use of FLASH [Magoč and Salzberg, 2011] code, the Linclust [Steinegger and Söding, 2018] algorithm and the MMseqs2 [Steinegger and Söding, 2017] code library. Code and binaries can be downloaded from GitHub at https://github.com/soedinglab/plass. The version used here for the benchmarks is GitHub commit 7571d37. A documentation for PenguiN can be found in section A.1 in the appendix.

2.2.2.9. Parallelization

Due to the large size of metagenomic samples, the assembly task can be computationally demanding and time-consuming. Modern CPUs offer the opportunity for efficient processing through parallelization on multicore processors. The different cores act thereby as independent units, but can communicate with each other "shared memory"). PenguiN supports multi-threading using OpenMP [Dagum and Menon, 1998] in almost all stages (overlap phase, extension phase, cycle detection, redundancy reduction). The parallelization is thereby mostly achieved by parallel processing of the input files in equal sized chunks enabled through the MMseqs2 database format. PenguiN makes use of the parallelization that was already implemented in the Plass, MMseqs2 and Linclust modules, but also all newly implemented modules support parallelization using OpenMP. By default, PenguiN utilizes all available CPU-cores, but the user can limit this with the --threads parameter. Additionally, SSE4.1/AVX2 instructions are used in the Linclust algorithm utilized for PenguiN's redundancy reduction step. For example, for the vectorized calculations of the reverse complements in the k-mer matching step, the Hamming distance pre-clustering step and the banded nucleotide alignment.

Further, similar to Plass, PenguiN can also be run on multiple homogeneous servers using the message passing interface (MPI). In this scheme, the execution of the program is split to multiple independent nodes ("distributed memory"), where each of them can use multiple cores. However,

access to the same file system from every node is important, as the folder for temporary files has to be shared between all nodes.

2.2.2.10. Parameter settings

PenguiN has multiple parameters which can be changed by the user and adapted to the input dataset. The most important parameters are probably the number of iterations of the assembly (--num-iterations), the sequence identity threshold for the overlap (--min-seq-id), the E-value threshold for overlaps (-e) and the minimal length to report a contig (--min-contig-len). Further, the sequence identity and coverage thresholds used for the Linclust algorithm to reduce the redundancy (--clust-min-seq-id and -–clust-min-cov) can be tuned. However, these should be done with caution as very strict parameters can result in high redundant assemblies, whereas very weak parameter settings can lead to a loss of diversity. Further, the new --cycle-check parameter, turning on the cycle detection in stage II, is crucial for the assembly as it prevents the assembly to cover the entire genome multiple times within a contig and should therefore not be turned off without a good reason.

Testing all combinations of parameters to find the best set was impractical and would probably highly depend on the length and quality of the input data, the coverage and the expected length of the full (viral) genomes. Therefore, most parameters were chosen based on the experience with Plass. However, the number of iterations for the protein and corresponding ORF assembly was reduced from 12 to 5 (default) due to the new logic of multiple extension within one iteration. I found 5 iterations on each of the two assembly stages (--num-iterations aa:5,nucl:5) working well for viruses ranging from 1 to 10 kbp and can also work for viruses up to 40 kbp depending on the number and length of the ORFs. However, to assemble much longer (viral) genomes more iterations might be necessary.

The behavior of different sequence-identity thresholds was analyzed in detail by Louis Kraft during his student internship. He found that 99% sequence identity on nucleotide-level gives high precision, independent of the values tried for the amino acid level, while keeping the sensitivity for all cutoffs. Higher sensitivity can be reached when lowering down the sequence identity values, but with a loss of precision for high cutoffs. For the sequence identity value on protein-level, 97% seems to be a reasonable choice. Notably, almost all parameters can be changed individually for the two guided assembly (stage I) and the nucleotide assembly (stage II) due to the new multi parameter logic, which I added to the MMseqs2 code library. However, usually, the default settings should already provide high-quality assemblies.

2.3. Test and Benchmark Design

A crucial task of my doctoral research, besides developing and implementing PenguiN, was to test and benchmark the resulting software tool in comparison to state-of-the-art and other

recently published assembly tools. In this section, I briefly summarize the benchmark setup. The respective results are then presented in the next section.

2.3.1. Computational resources

The benchmark was performed on the GWDG HPC cluster (www.gwdg.de) using individual nodes with two Intel Xeon E5-2640v3 processors at 2.6 GHz totaling 16 cores and 128 GB RAM for each assembly run. The operating system used was Scientific Linux release 7.9 (Nitrogen). Before each assembly run, sequencing read data was always copied to the local SSD ($\sim$ 400 GB). Likewise, paths for output and possible temporary files were also set to the local disk. All assembly runtimes were limited to 10 days per sample.

2.3.2. Choice of assembly software and parameter settings

At the time of the design of the project, there were basically two types of assemblers applied on viral metagenomic data: (general) metagenomic and metatranscriptomic assemblers, which were initially designed for mixtures of more divergent microbial genomes [Sutton et al., 2019] and viral specific assemblers, designed for the application on patient samples. In the scope of my doctoral research, I compared PenguiN with both groups. Megahit [Li et al., 2015], metaSPAdes [Nurk et al., 2017] and rnaSPAdes [Bushmanova et al., 2019] from the first group and SAVAGE [Baaijens et al., 2017], IVA [Hunt et al., 2015] and VICUNA [Yang et al., 2012] from the second group.

During the work on this project, three further assemblers were published, which consider viral genomes in metagenomic samples: metaviralSPAdes [Antipov et al., 2020], which is designed to assemble DNA viruses (cyclic and linear) from metagenomic samples, rnaviralSPAdes [Meleshko et al., 2022], which was published together with its more specialized variant coronaSPAdes and is tailored to assemble RNA viruses from transcriptome, metatranscriptome, and metavirome datasets, and Haploflow [Fritz et al., 2021], which aims for strain-resolved assembly and was initially tested on patient samples as well as metagenomic samples identical to clinical isolates. Consequently, all three tools were added later to the benchmark analysis as well.

The respective software versions of the all assembly tools used in the benchmark are: PenguiN (GitHub commit 7571d37), Megahit (v1.2.9), metaSPAdes/rnaSPAdes/metaviralSPAdes/rnaviralSPAdes (v3.15.2), SAVAGE (v0.4.2), IVA (v1.0.8), VICUNA (v1.3), Haploflow (v0.1).

Each assembler was run with default parameters, except for the minimum contig length and the CPU thread parameter settings, which was uniformly set to utilize all 16 available cores. Haploflow was the only tool that did not support multi-threads. The minimum contig length was set to 500 bp or 1000 bp depending on the dataset for all assemblers that provide such a filter option. For all others, the final assembly file was filtered subsequently with the same minimum length threshold. Possible minor deviations from these settings, which were necessary

for some assemblers to be applicable to some datasets, are noted in the respective subsections of the results (section 2.4). All tools were provided with paired-end read files except for Haploflow, as Haploflow does not provide the possibility to take separate paired-end read files as input. In order to run it, I appended both ends to a single file according to the same strategy used in the Haploflow publication benchmark [Fritz et al., 2021]. For SAVAGE, it was necessary to set the additional --revcomp flag to specify the orientation of the reads.

2.3.3. Choice of test and benchmarking datasets

On real metagenomic samples no ground truth is available. Therefore assemblers can only be assessed using reference-free assembly statistics such as assembly size, read mapping rate, N50, etc. This heavily limits the assessment of the performance of the assembler in terms of completeness of the recovered genomes. Furthermore, it also limits the ability to detect wrongly assembled sequences (false positives). Therefore, simulated datasets with a known composition are generally the better choice to start with. Within my doctoral research, I evaluated PenguiN on two simulated datasets. The first simulated dataset was generated from three *human rhinovirus* genomes (HRV), the second dataset comprised of 2550 HIV-1 genomes. HRV was chosen because it is a well-studied virus with a simple genome of about 7 kbp that only contains a single open reading frame (ORF) ($\sim$6.5 kbp). HIV was chosen as it provides an extensive data source for natural genetic variation resulting from its fast replication and high mutation rate, as well as recombination events due to multi strain infections [Bbosa et al., 2019; Hemelaar et al., 2006; Leye et al., 2013]. It therefore allows simulating a complex dataset with natural variation and known reference sequences. Further, HIV genomes were already commonly used in several previous studies to evaluate performance of assembly tools [Baaijens et al., 2017; Chen et al., 2018; Fritz et al., 2021]. To analyze the effect of different coverage levels, I simulated three different sets from the HIV-1 2550 genomes varying in coverage (1-fold, 10-fold, 100-fold).

Additionally, I also tested PenguiN on real HiSeq 2500 Illumina sequencing data (comprised of 2×300 bp reads) obtained from a lab mixture of six *Caudovirales* genomes used in a previous study [Warwick-Dugdale et al., 2019], referred to as a mock community. Members belonging to the order of *Caudovirales*, tailed dsDNA phages, are of special interest as they account for most known bacteria infecting viruses ($>96\%$) [Ackermann and Prangishvili, 2012] and are highly representative in many samples obtained in previous metagenomic studies [Nayfach et al., 2021b; Yang et al., 2019]. Finally, I used a dataset comprised of 82 real metatranscriptomic samples from activated sludge and aquatic environments, which were previously used to detect ssRNA phage genomes from these two environments [Callanan et al., 2020]. Members of ssRNA phages, making up the *Fiersviridae* (since 03/2021; previously *Leviviridae* [Walker et al., 2021]) family, have small positive stranded RNA genomes of 3.5–4.5 kbp encoding three or four proteins [Chamakura et al., 2020; Tars, 2020] and have been used as a model to understand various fundamental processes in virology and molecular biology [Gytz et al., 2015; Lodish, 1968; Tars, 2020].

The details of the read simulations or downloaded datasets, as well as the description of read pre-processing and assembly for each dataset, can be found together with the results within section 2.4.

2.3.4. Evaluation tools and metrics

For the first three datasets, reference genomes were available, so I could use the reference-based third-party tool MetaQUAST [Mikheenko et al., 2016a] to evaluate the quality of the assembled contigs. MetaQUAST is commonly used to assess metagenomic assembly quality [Meyer et al., 2021; Sczyrba et al., 2017; Sutton et al., 2019] and provides several metrics including genome fraction, number of misassemblies, number of mismatches per 100 kbp, duplication ratio, N50, NGA50 and others. Further, it reports read statistics such as mapping rates to the contigs using the short read aligner bwa [Li, 2013], if read files are specified.

In the scope of this thesis, I used MetaQUAST version 5.0.2 with a minimum contig length of 1000 bp, the --unique-mapping flag and default parameters otherwise. The --unique-mapping flag ensures that for every contig only one alignment is considered, if multiple equally well alignments exist. It is commonly used in metagenomic assembly benchmark studies [Meyer et al., 2021; Sczyrba et al., 2017; Sutton et al., 2019]. By default, MetaQUAST performs all reference-based metric calculations with a sequence identity cutoff of 95%. Alignments of contigs to references are thereby only counted if their percent identity lies above this threshold.

Further comparison of a subset of assemblers (PenguiN, Megahit, metaSPAdes, rnaSPAdes, rnaviralSPAdes) were carried out on the HIV-1 dataset using MMseqs2 [Steinegger and Söding, 2017] to search assembled contigs throughout the set of reference genomes, and vice versa, for the calculation of sensitivity and precision values similar to [Steinegger et al., 2019]. The details of this analysis are described in section 2.4.2.

For the real environmental metatranscriptomic data, I searched for all ssRNA phage sequences in the resulting assemblies following the analysis done in [Callanan et al., 2020], and compared the results to those obtained in Callanan's analysis. Additionally, I compared PenguiN's results to those from other assemblers in terms of the number of assembled phage sequences (partial and complete), determined the overlap of the sets of completely assembled ssRNA phage genomes using MMseqs2 [Steinegger and Söding, 2017] and investigated the reliability of PenguiN's assembled genomes in various downstream analyses, examining the genome lengths, the genome architecture, the core protein associations etc.

2.4. Evaluation and Results

In this section, I show the performance of PenguiN on the four mentioned datasets (two simulated dataset, one in vitro mixture, one real metatranscriptomic dataset) and compare it with those of nine other assemblers. In each of the subsections (2.4.1-2.4.4), I first describe the characteristics

of the simulated or downloaded dataset and subsequently present the assembly results. Finally, I evaluate PenguiN's runtime and memory usage in comparison to the other assemblers on these datasets (section 2.4.5).

2.4.1. Performance on simulated error-free reads

For a proof of concept, PenguiN was first tested on a synthetic dataset from error-free reads of three *human rhinovirus* strains (HRV in silico mixture). For this purpose, I downloaded the genomes of *Rhinovirus A1 strain 5Q1*, *Rhinovirus A1 strain 7A2* and *Rhinovirus A1B strain 12O2* (Accession No.: MF973193.1, MF973194.1, MN749156.1) from the NCBI GenBank (www.ncbi.nlm.nih.gov/genbank/) [Sayers et al., 2019] with ANI values ranging from 92 to 95.5%, mixed them in a proportion of 4:2:1 and simulated 2×150 bp overlapping paired-end reads (insert size range 220-280 bp) using randomreads.sh from the BBmap software suite (version 38.71) [Bushnell, 2014]. The simulated coverage was set to 50, resulting in theoretical read depths of 200:100:50.

Afterwards, reads were assembled using PenguiN, Megahit, metaSPAdes, metaviralSPAdes, rnaSPAdes, rnaviralSPAdes, SAVAGE, IVA, VICUNA, and Haploflow. Thereby, metaviralSPAdes did not produce any final contigs and was therefore excluded from the subsequent analysis. The assembly quality was then evaluated using MetaQUAST (see Fig. 2.4). PenguiN performed well in all metrics. It assembled three contigs, covering all three strain genomes with a single contig each (Fig. 2.5). In comparison, the general metagenomic and metatranscriptomic assemblers (Megahit, metaSPAdes, rnaSPAdes) resulted in more fragmented assemblies (higher number of contigs and lower NGA50 value per genome) and missed larger parts of at least one of the genomes. Megahit, for example, could still recover 92% of the total genome fraction, but the strain MN749156.1, which was simulated with the lowest abundance, was only recovered by 80%. metaSPAdes recovered 60% of the total genome fraction including only 20% of MN749156.1.

The multiple sequence alignment (MSA) of the three reference genomes shows that the three HRV genomes mostly differ by SNPs distributed over the whole sequences (Fig. 2.6). However, larger parts are identical between two or all three genomes. Megahit and metaSPAdes seem to have problems exactly at those regions, as both achieved fragmented contigs on the edges of the shared regions. I believe that this is due to their use of de Bruijn graphs, since they can not be overlaid when splitting the reads in shorter k-mers and lead therefore to ambiguous paths in the graph producing fragmented contigs. This indicates a possible advantage of assemblers using whole reads such as PenguiN or SAVAGE that can utilize the co-occurrence of mutations within one read length. An exception among the de Bruijn graph assemblers seems to be Haploflow, which assembled three contigs covering the three genomes, probably because it can take advantage of the different coverage values within its flow algorithm [Fritz et al., 2021]. However, Haploflow yielded a significantly higher number of mismatches per 100 kbp than SAVAGE and PenguiN, indicating an inadvertent mix of SNPs from the three genomes in the assembled contigs due to the absence of sequencing errors in the simulated reads. In total, the viral specific assem-

Worst Median Best ☑ Show heatmap

Genome statistics	penguin	megahit	metaspades	rnaspades	rnaviralspades	savage	iva	vicuna	haploflow
Genome fraction (%)	99.995	92.269	60.392	85.473	69.634	97.388	27.929	33.511	97.983
Duplication ratio	1.009	1.003	1.009	1.289	1.005	1	1.055	1.005	1.01
Largest alignment	7137	6884	6295	7096	6821	7087	3527	7097	7095
Total aligned length	21 174	19 599	12 898	23 291	14 818	20 622	6227	7097	20 748
NGA50	...	...	...	...	...	...	...	...	...
LGA50	...	...	...	...	...	...	...	...	...
Reads mapping									
Mapped (%)	100	99.42	92.78	100	98.79	100	87.59	99.93	100
Properly paired (%)	100	98.18	89.87	100	97.62	99.31	84.69	99.93	98.52
Singletons (%)	0	0.38	1.2	0	0.46	0	1.82	0	0
Misjoint mates (%)	0	0.73	1.49	0	0.63	0.69	0.67	0	1.48
Avg. coverage depth	115	124	174	105	163	119	336	344	117
Coverage >= 1x (%)	99.15	100	100	99.8	100	100	100	100	99.52
Misassemblies									
# misassemblies	0	0	0	0	0	0	0	0	0
Misassembled contigs length	0	0	0	0	0	0	0	0	0
Mismatches									
# mismatches per 100 kbp	0	40.95	117.3	1331.57	0	0	236.73	746.9	515.71
# indels per 100 kbp	0	0	0	27.63	0	0	0	14.09	0
# N's per 100 kbp	9.36	0	0	0	0	0	0	0	0
Statistics without reference									
# contigs	3	5	5	7	4	3	3	1	3
Largest contig	7138	6884	6295	7127	6821	7087	3542	7128	7126
Total length	21 357	19 599	12 898	23 322	14 818	20 622	6242	7128	20 965
Total length (>= 1000 bp)	21 357	19 599	12 898	23 322	14 818	20 622	6242	7128	20 965
Total length (>= 10000 bp)	0	0	0	0	0	0	0	0	0
Total length (>= 50000 bp)	0	0	0	0	0	0	0	0	0

Figure 2.4.: Assessing assembly quality for PenguiN and eight other assemblers on the HRV in silico mixture using MetaQUAST. Cells differing from the median are colored. All statistics are based on contigs of size ≥ 1000 bp, unless otherwise noted. Metrics that depend on the reference lengths (e.g. NGA60, LGA50) are not calculated for the combined reference. A more detailed report per reference can be found in Fig. A.2 in the appendix.

blers split up into two groups. Whereas the strain-aware assemblers SAVAGE and Haploflow could recover all three genomes, IVA and VICUNA assembled only one of the strains due to their design as viral consensus assemblers (Fig. 2.4, Fig. 2.5).

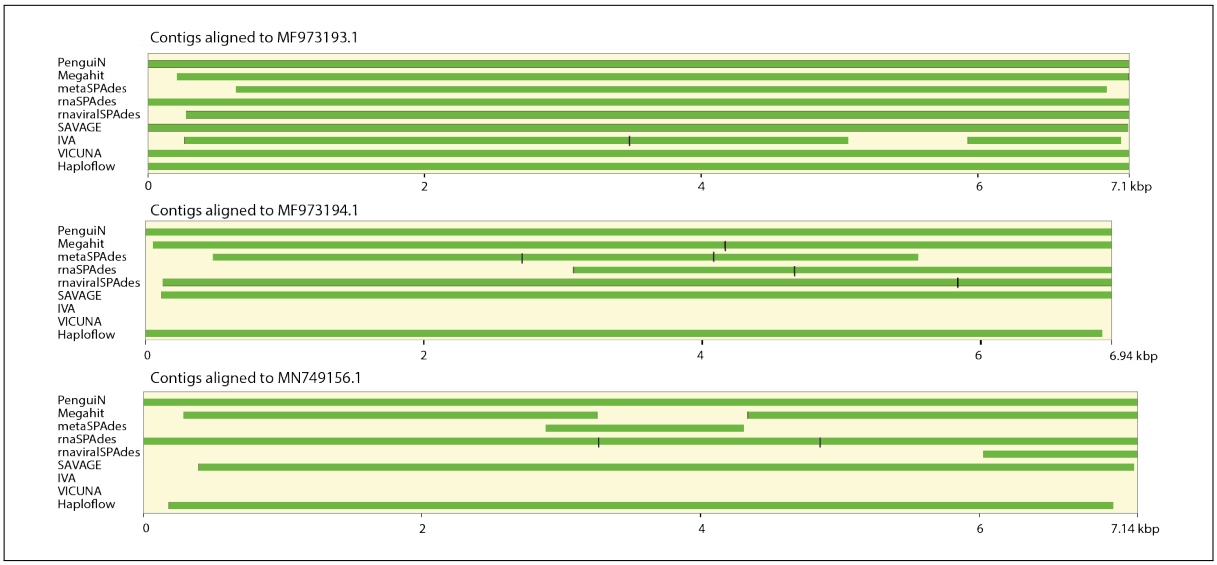


Figure 2.5.: Contigs aligned to the three reference genomes used for the HRV in silico mixture. Visualization of contig alignments was created with Icarus [Mikheenko et al., 2016b], which is automatically called within the MetaQUAST pipeline, if reference genomes are provided. Black vertical lines indicate overlapping contigs.

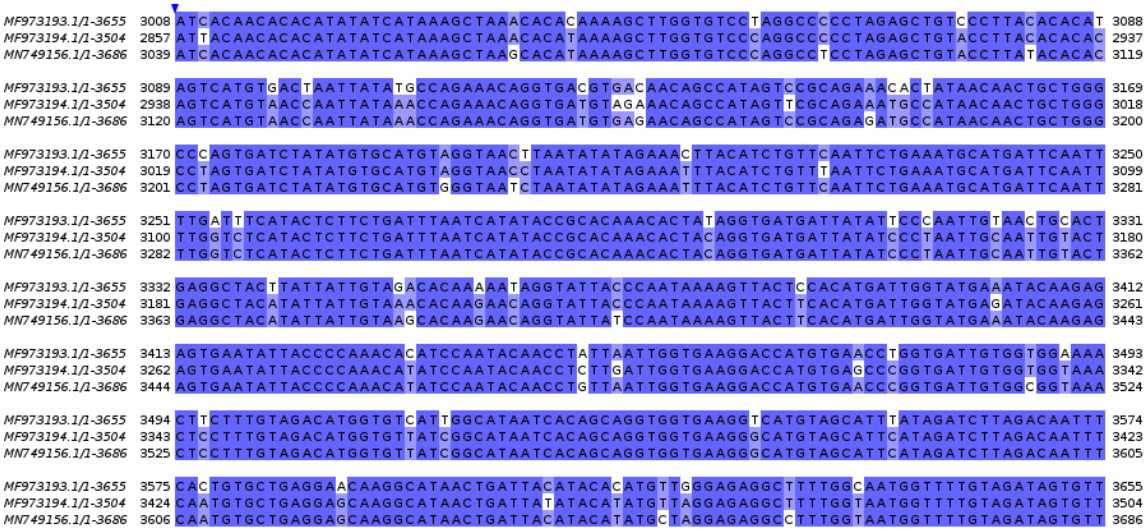


Figure 2.6.: Extract of the multiple sequence alignment (MSA) of the three rhinovirus reference genomes used to simulate the HRV in silico mixture. The whole MSA was computed using MAFFT [Katoh et al., 2002] and visualized using Jalview (version 2.11.1.4) [Waterhouse et al., 2009]. The color scheme indicates the percentage of the residues in each column that agree with the consensus sequence. Thus, the blue color marks columns where all three nucleotides match, the lighter blue color marks columns where two of three match, the white color marks unique nucleotides. The three sequences mainly differ by SNPs that can be found all over the sequence, but there are also large identical parts.

2.4.2. Performance on a highly diverse strain mixture

Next, I tested the performance of PenguiN on a highly diverse strain mixture. To this end, I used 2550 HIV-1 genomes, which I obtained by searching for all complete HIV-1 genomes in the NCBI database (accessed 07/2019) using the following search string: ("Human immunodeficiency virus 1"[Organism] OR hiv1[All Fields]) AND complete genome[All Fields]). I simulated three datasets of 2×150 bp error-free reads with a mean genome coverage of 1x, 10x and 100x again using randomreads.sh from the BBmap software suite (*HIV1 in silico mixtures*). The assembly quality was evaluated (i) using metrics from MetaQUAST [Mikheenko et al., 2016a] and (ii) using MMseqs2 [Steinegger and Söding, 2017] to compute precision and sensitivity values as previously done [Steinegger et al., 2019]. The details are described in the following. However, before presenting the results, I shortly describe the peculiarities of the dataset construction, requiring a brief review of the HIV-1 genome architecture first.

HIV-1 is a single-stranded RNA virus which evolves rapidly, leading to extreme genetic diversity [Bbosa et al., 2019], and it therefore provides a good source for constructing a benchmark set with a high degree of natural variation. Comparing the average nucleotide identity (ANI) of the 2550 HIV1 genomes showed values mostly in the range from 85-95% (see Fig. 2.7). Within the ANI distribution in Fig. 2.7, two peaks can be made out, which can be attributed to inter- and intra- subgroup identities [Désiré et al., 2018]. The HIV1 genome is approximately 9.7 kbp and flanked at both ends by long terminal repeats (LTRs) that can interact, leading to circularized forms containing one or two copies of the viral LTR (1-LTR circles, 2-LTR circles) [Craigie and Bushman, 2012; Maldarelli et al., 2016]. All downloaded 2550 HIV-1 genomes were reported as "complete genome" in the NCBI database, but their LTRs were reported in an inconsistent manner (without, with one LTR or two LTRs, etc.). Because of that, it was not clear what result an assembler should ideally produce (genome with one LTR or two LTRs, etc.) and circularized contigs would heavily hamper the comparison of the different assemblers (rearrangement of the contig start positions would be necessary). Therefore, I consistently circularized all genomes by removing one of the repetitive regions, if two were present, and doubling the sequences afterwards before simulating reads (Fig. 2.8). In this way, simulated reads from circular genomes were provided consistently as input for all assemblers.

Subsequently, I assembled the reads using PenguiN and the above-mentioned nine other assembly tools. Notably, metaSPAdes and metaviralSPAdes needed to be run with the option --only-assembler set, as otherwise the calls finished abnormally due to an error thrown by BayesHammer [Nikolenko et al., 2013], which is internally called for the read error correction step. The reason was not investigated further. Table 2.1 shows the number of contigs with a minimum length of 1000 bp produced by each tool. In general, the viral specific assemblers provided much lower numbers of contigs than the metagenomic assemblers or did not finish. For the 10x coverage set, Haploflow finished without producing a final assembly file (without any error message). It seems that no contigs greater than 1000 bp were assembled. Therefore, it was excluded from the analysis for this subset. Furthermore, for the 100x coverage set, Haploflow was terminated

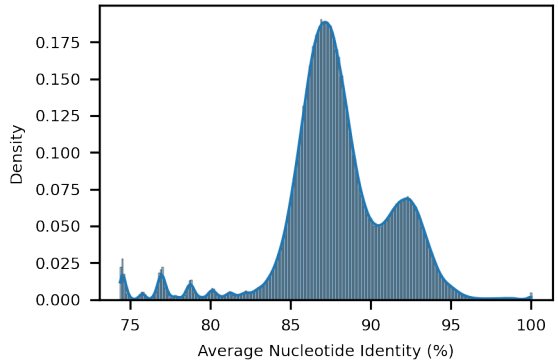


Figure 2.7.: Average Nucleotide Identity (ANI) distribution for the pairwise comparison between the 2550 HIV genomes. ANI values were estimated using fastANI version 1.1 [Jain et al., 2018]. The distribution misses 75 000 ($\sim 1\%$) pairwise ANI values. They could not be estimated by fastANI, which can fail for ANI values smaller than 80%.

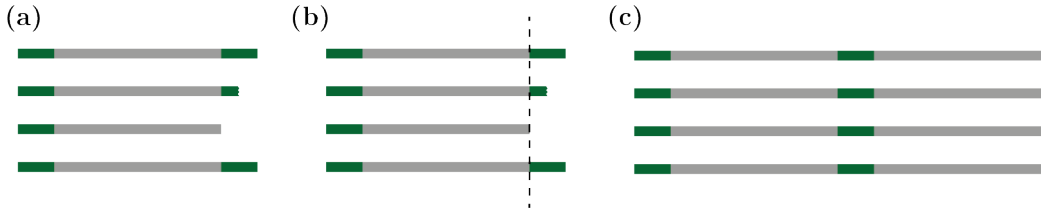


Figure 2.8.: Illustration of the reference genome preparation for the 2550 HIV1 in silico mixture. (a) The HIV1 genomes are reported in an inconsistent manner in the database (LTR colored in green). (b) For each genome, the repetitive region is identified by searching the first half of the genome against the second half using MMseqs2 and subsequently cut. (c) Afterwards, the cut genomes are doubled to circularize the genomes consistently. Reads are then simulated from these *wrapped* versions of the genomes.

	1x	10x	100x
PenguiN	**1139**	**6205**	5066
Megahit	74	2289	3365
metaSPAdes	20	670	2254
metaviralSPAdes	0	1	149
rnaSPAdes	309	5969	**7097**
rnaviralSPAdes	49	1574	2082
SAVAGE	2	-	-
IVA	1	11	43
VICUNA	5	22	983
Haploflow	1	-	-

Table 2.1.: Number of contigs (≥ 1 kbp) per assembler for the three subsets (1-fold, 10-fold, 100-fold) of the 2550 HIV1 in silico mixture. The highest value per column is marked.

after 10 days without any result. SAVAGE also did not complete on either of these sets within 10 days. Therefore, Haplfolow and SAVAGE were only considered for the 1x coverage set.

2.4.2.1. Analysis using MetaQUAST

As above, MetaQUAST was used with the --unique-mapping flag, a minimum contig length of 1000 bp, and default parameters otherwise to evaluate the assemblies. However, this time I skipped the integrated read mapping as it did not finish in reasonable time (>10 days for the 100-fold coverage set). Table 2.2 shows selected MetaQUAST metrics, the full MetaQUAST reports for all three subsets are provided in Fig. A.3 in the appendix.

Overall, I obtained a 2 to 5-fold increase in the total assembly size of PenguiN compared to the next best assembly size achieved by rnaSPAdes, whereas the rate of misassemblies were similar or even lower. E.g. compared to rnaSPAdes at 1x coverage, PenguiN has a 4.5 times larger assembly but only 2.5 times as many misassemblies. At coverage 10x and 100x, PenguiN even has a lower number of misassemblies while having a two times larger assembly. Also in comparison to Megahit or metaSPAdes, PenguiN showed comparable misassembly rates. This hints, that PenguiN's larger assembly size does not come with a loss of precision in comparison to the metagenome assemblers. The duplication ratio of PenguiN's assembly was slightly higher than for most of the other assemblers (except for rnaSPAdes). I believe this is due to the less aggressive settings of PenguiN's internal redundancy reduction (using Linclust at 97% sequence identity and 99% minimum coverage of the shorter sequence) to avoid clustering of different strains. However, the duplication level was still reasonable (ranging from 1.12 to 1.32 for the combined reference).

Fig. 2.9 displays genome fraction and the NGA50 values for the individual genomes reported by MetaQUAST. Thereby, the genome fraction measures the total number of aligned bases in the reference divided by the genome size. For all three sets (1x, 10x, 100x coverage) PenguiN recovered the highest fraction with a median of 81.20% for 10x coverage and 94.25% for 100x coverage, which were substantially higher than the ones of the next best tools (rnaSPAdes: 28.16% and 34.58%, Megahit: 12.09% and 31.16%). Further, PenguiN was the only tool that could recover target genomes by more than 90% genome fraction on the 1x coverage set, which was achieved 20 genomes. On the 10x coverage set, PenguiN recovered 1006, and on the 100x coverage set 1408 genomes by more than 90% genome fraction, representing an approximately 40-fold increase compared to the next best tools (rnaSPAdes: 24 and 36, Megahit 3 and 11).

Analyzing the NGA50, as depicted in Fig. 2.9b, also complements this observation. While almost all assemblers can gain from more coverage, most assemblers except for PenguiN have either fragmented assemblies or do not cover the reference genomes with $\geq$ 50% (counted as NGA50=0). PenguiN's much higher NGA50 value per reference shows that PenguiN assembles longer and more complete contigs.

The assemblers SAVAGE (de novo mode) and Haploflow, which aim for strain-resolved assemblies and which performed comparable to PenguiN on the three *human rhinovirus* in silico mixture (section 2.4.1), are basically not applicable on this data. Providing SAVAGE with a reference might help [Deng et al., 2021] but was not tested in the scope of this work as it would

Table 2.2.: Selected metrics of the MetaQUAST analysis for the assemblies of the three subsets (1-fold, 10-fold, 100-fold) of the 2550 HIV1 in silico mixture. All statistics are based on contigs of size ≥ 1000 bp. The best value per column is bold. Haploflow and SAVAGE did not produce results for the 10-fold and 100-fold subset within 10 days.

(a) 1x coverage set

	# of contigs ≥ 1 kbp	Total length (bp)	Genome fraction (%)	# Misassemblies	Duplication ratio
PenguiN	1139	2 087 455	**8.147**	282	1.12
Megahit	74	92 163	0.387	69	1.042
metaSPAdes	20	23 795	0.101	2	1.026
metaviralSPAdes	0	-	-	-	-
rnaSPAdes	309	453 490	1.645	112	1.203
rnaviralSPAdes	49	63 753	0.275	10	1.01
SAVAGE	2	2234	0.01	**0**	**1**
IVA	1	2491	0.011	**0**	0.999
VICUNA	5	13 479	0.058	3	1.01
Haploflow	1	1084	0.005	**0**	1.01

(b) 10x coverage set

	# of contigs ≥ 1 kbp	Total length (bp)	Genome fraction (%)	# Misassemblies	Duplication ratio
PenguiN	6205	20 006 079	**66.46**	772	1.316
Megahit	2289	3 533 275	14.903	674	1.036
metaSPAdes	670	971 940	4.18	126	1.016
metaviralSPAdes	1	9167	0.04	**0**	1.014
rnaSPAdes	5969	9 978 913	29.669	2362	1.467
rnaviralSPAdes	1574	2 370 527	10.324	68	**1.004**
SAVAGE	-	-	-	-	-
IVA	11	31 319	0.127	10	1.017
VICUNA	22	40 812	0.176	12	1.014
Haploflow	-	-	-	-	-

(c) 100x coverage set

	# of contigs ≥ 1 kbp	Total length (bp)	Genome fraction (%)	# Misassemblies	Duplication ratio
PenguiN	5066	21 076 302	**71.941**	612	1.281
Megahit	3365	6 897 024	29.365	318	1.027
metaSPAdes	2254	3 656 910	15.748	234	1.013
metaviralSPAdes	149	513 966	2.044	**2**	1.099
rnaSPAdes	7097	12 654 174	35.058	3080	1.573
rnaviralSPAdes	2082	3 347 222	14.569	72	**1.005**
SAVAGE	-	-	-	-	-
IVA	43	151 664	0.533	83	1.058
VICUNA	983	1 504 140	6.505	146	1.01
Haploflow	-	-	-	-	-

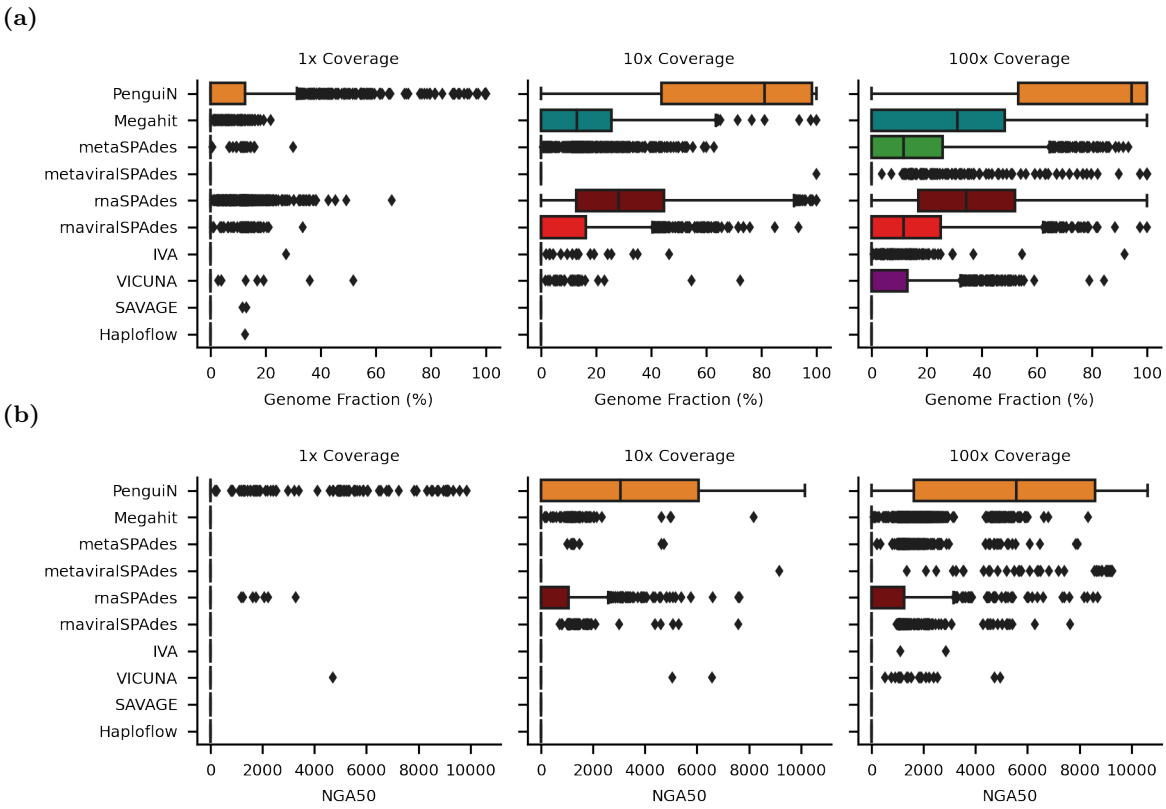


Figure 2.9.: Assessing assembly results for individual genomes with MetaQUAST for the three subsets (1-fold, 10-fold, 100-fold) of the 2550 **HIV1 in silico mixture.** (a) Genome fraction of the individual reference genomes assembled by each assembler for all genomes, (b) NGA50 statistics per reference genome. NGA50 is analogous to NG50, the length for which the collection of all aligned blocks of that length or longer covers at least half the reference genome. If less than half of a reference genome is covered, the NGA50 for that genome is not defined, these values are set to zero here. The higher the genome fraction and NGA50, the better is the assembly quality. Within the boxplot, vertical lines indicate the median, while the boxes represent the quartiles (25, 75) with 1.5 times extension by the whiskers, outliers are shown as diamonds.

not be the realistic use-case in a metagenomic context. Haploflow instead seems to prune its de Bruijn graph structures too drastically, and therefore cannot make use of its flow algorithm in the second part for the strain-resolved assembly. IVA and VICUNA also performed very poorly, obtaining the lowest genome fractions recovered with a high number of mismatches. Consequently, these viral specific assemblers were excluded from the subsequent analysis. I also excluded metaviralSPAdes due to its poor performance and only continued with the other SPAdes based assemblers.

2.4.2.2. Analysis using MMseqs2 search

Complementary to the analysis using MetaQUAST, I evaluated the per-base sensitivity and per-base precision at different sequence identity cutoffs for the assemblies from PenguiN, Megahit, metaSPAdes, rnaSPAdes and rnaviralSPAdes.

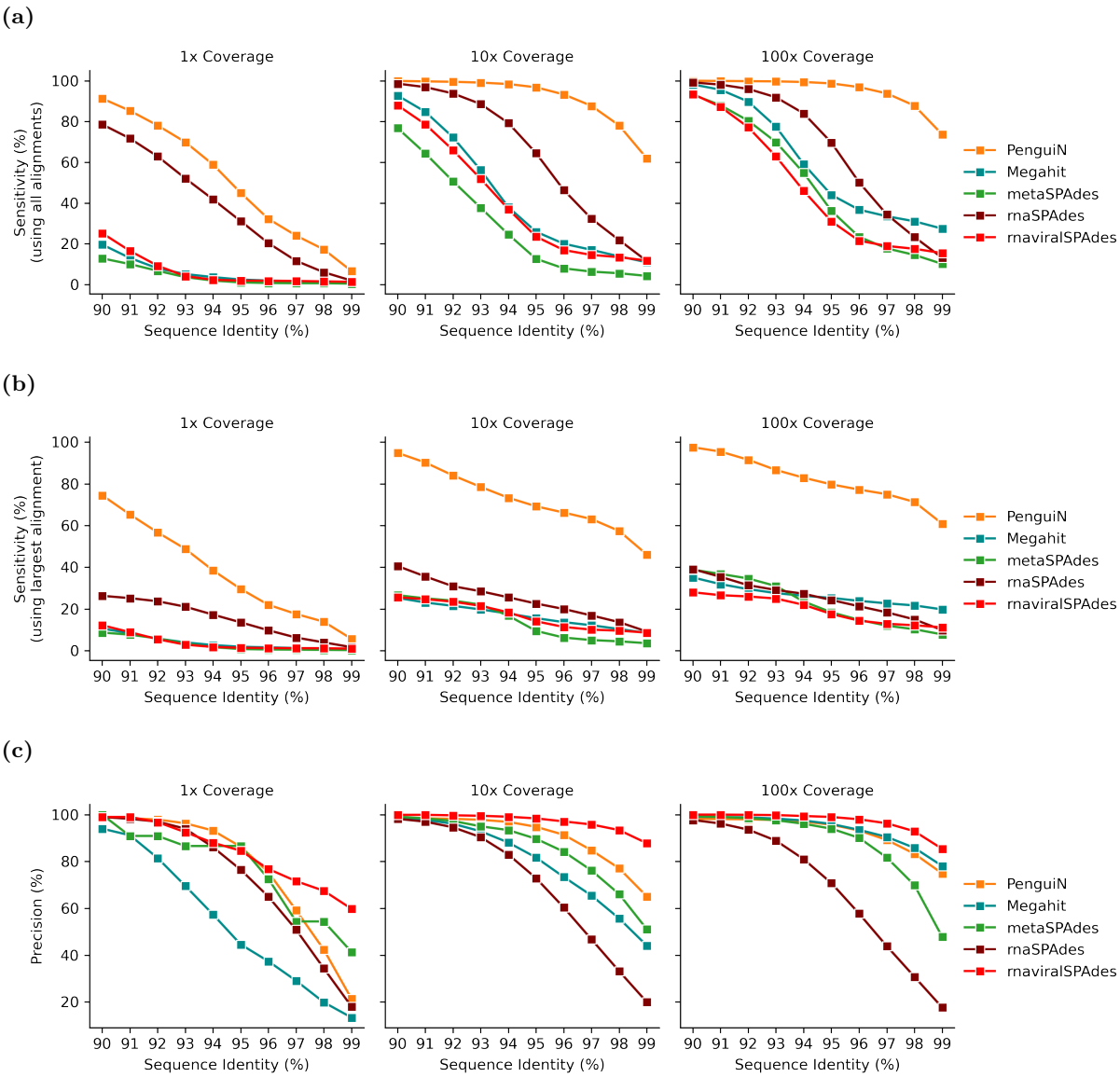


Figure 2.10.: Sensitivity and precision of contigs assembled from the HIV1 in silico mixture. (a) Assembly sensitivity computed from all alignments, (b) Assembly sensitivity only considering the largest alignment, (c) Assembly precision. Each metric is shown for all three subsets (1-fold, 10-fold, 100-fold) from left to right.

Sensitivity and precision calculation Sensitivity and precision were computed similar to [Steinegger et al., 2019]. For the sensitivity, I searched with the set of reference genomes through the assembled contigs using MMseqs2 with options -a -s 5.7 --max-seqs 500000 --min-seq-id 0.89 --strand 2 --search-type 3 --max-seq-len 1000000 and subsequently filtered the resulting alignments with a minimum sequence identity threshold between 90% and 99%. Sensitivity was then defined as the fraction of the reference sequences that can be covered by contigs with a sequence identity of at least that threshold (total count of aligned nucleotides divided by total length of reference genomes). Thereby, I considered two ways, either considering all alignments or only the longest alignment for each reference genome. The latter approach corresponds to

the sensitivity definition used in [Steinegger et al., 2019]. Precision was calculated by searching with the contigs through the set of reference genomes, using MMseqs2 with options -a -s 5.7 --max-seqs 5000 --min-ungapped-score 100 -a --min-seq-id 0.89 --strand 2 --search-type 3 --max-seq-len 10000000. For each contig, only the longest alignment was considered. Precision was then defined as the fraction of contigs that are aligned to the reference genomes (total count of aligned nucleotides divided by total length of assembled contigs).

Fig. 2.10a shows the sensitivity values computed from all alignments at different sequence identity cutoffs (X-axis). The sensitivity is quite similar for Megahit, metaSPAdes and rnaviralSPAdes over all sequence identity cutoffs, whereas rnaSPAdes assembled many more nucleotides correctly below X=98%. However, PenguiN always obtained the highest sensitivity values. Thereby, the difference is larger at higher sequence identity values. At the highest sequence identity cutoff (X=99%) PenguiN assembled approximately 3-5 times more nucleotides correctly than the next best tool (1x coverage: PenguiN 6.6%, rnaSPAdes 1.9%; 10x coverage: PenguiN 61.9%, rnaSPAdes: 11.5%; 100x coverage: PenguiN: 73.7%, Megahit 27.4%).

In contrast, at X=90% the sensitivity values of all tools were much more similar except for the low coverage set. E.g. for 100x coverage, all tools reached high sensitivity ranging from 93.5% to 100% for X=90%. However, at X=95%, the sensitivity dropped already to 69.7% for rnaSPAdes and 44% for Megahit, whereas PenguiN still reached 98.7%. This indicates that PenguiN could recover more strains correctly, whereas there is a lack of strain variation in the assemblies of the other assemblers, which is discussed in more detail in section 2.5.2. Only for coverage 1x, the sensitivity also significantly dropped for PenguiN (from 91.3% at X=90% to 6.6% at X=99%), indicating that reads from different strains were combined here too. This is probably due to the low coverage value, as a simulated mean coverage of 1 leads to incomplete coverage of the full viral genomes in the sequencing reads, making a mix of variants necessary to recover the full length.

If only the longest instead of all alignments is considered for the sensitivity calculation, equivalent to the sensitivity definition used in [Steinegger et al., 2019], the difference between PenguiN and the other assemblers is even greater for all sequence identity cutoffs (Fig. 2.10b). Altogether, the per-base sensitivity based on the largest alignment only was much smaller in comparison to the per-base sensitivity on all alignments for X=90-95% for all assemblers except PenguiN. This shows that PenguiN's high sensitivity relies mainly on the largest alignment only, whereas the sensitivity values of the other assemblers were composed of multiple alignments, indicating a much higher level of fragmentation within their assemblies. This is consistent with the NGA50 values reported by MetaQUAST, described in the previous section (Fig. 2.9).

In terms of precision, PenguiN performs similar to the other assemblers (Fig. 2.10c). Only rnaviralSPAdes achieved higher precision values on all three coverage sets, but with fewer contigs (and much smaller sensitivity values). For all three coverage sets (1x, 10x, 100x) 98 to 99% of PenguiN's contigs could be aligned back to the reference genomes with sequence identity values $\geq$90%. For $X = 99\%$ still 21.4% (for coverage 1x), 65%, (for coverage 10x) and 74% (for coverage

Phage	Family	Genome Length (kbp)	NCBI Accession
PSA-HS2	*Siphoviridae*	38.2	KF302036.1
Cba phi18:1	*Siphoviridae*	39.2	NC_021790.1
PSA-HP1	*Podoviridae*	45.0	KF302037.1
Cba phi38:2	*Myoviridae*	54.0	KC821629.1
Cba phi38:1	*Podoviridae*	72.5	NC_021796.1
PSA-HM1	*Myoviridae*	129.4	KF302034.1

Table 2.3.: Characteristics of the members included in the mock community produced in [Warwick-Dugdale et al., 2019].

100x) could be aligned to the reference genomes. This supports the above formulated hypothesis that PenguiN's much higher sensitivity values do not come with a loss of precision in comparison to the other assemblers. It can be seen that, due to its two-stage approach, PenguiN is able to mix variants to boost the sensitivity when the coverage is low, but at the same time is able to produce much longer and highly accurate assemblies when the coverage is sufficient. Thereby, it benefits from it's full-read overlap approach, which can distinguish closely related variants more accurately than the de Bruijn graph assemblers (as also seen for the HRV dataset).

2.4.3. Evaluation on a mock community

In the previous two sections, I showed that PenguiN performs well on simulated data. Before applying PenguiN on real environmental samples, where no ground truth is available, I tested PenguiN on a lab generated mock community of six marine *Caudovirales* (in vitro mixture) [Warwick-Dugdale et al., 2019]. Hence, the effect on the assembly quality due to the challenges of real sequencing data could be investigated. Originally, short-read sequencing on a HiSeq 2500 (Illumina Inc.) and long read sequencing on a MinION flow cell (Oxford Nanopore Technologies) was performed by Warwick-Dugdale et al. Here, I only used the set of Illumina short sequencing reads, comprising 5 642 957 paired-end reads of 2×300 bp with a total base count of 2.8 billion. The sequencing reads were downloaded from the European Nucleotide Archive under the project accession number PRJEB27181 (https://www.ebi.ac.uk/ena/browser/view/PRJEB27181). Following [Warwick-Dugdale et al., 2019], I pre-processed them using Trimmomatic (version 0.39) [Bolger et al., 2014] to remove adapters and low-quality reads and bbnorm (https://sourceforge.net/projects/bbmap/) to normalize to ∼100-fold coverage, yielding 133 571 paired-end reads with ∼61 million base-pairs. In principle, normalization is not necessary to run PenguiN, but it sped up the assembly.

The six *Caudovirales* included in the mock community represent the three main families of dsDNA bacteriophages (see Table 2.3). As the genome sizes of the six *Caudovirales* ranges from 38.2–129.4 kbp, the expected contig sizes were much larger compared to the HRV or HIV datasets. This was compensated by increasing the number of iterations for PenguiN's assembly process using the option --num-iterations aa:5,nucl:7 (instead of the default values aa:5, nucl:5).

Fig. 2.11 shows how the length of the largest contig for each reference genome increased with the numbers of iterations.

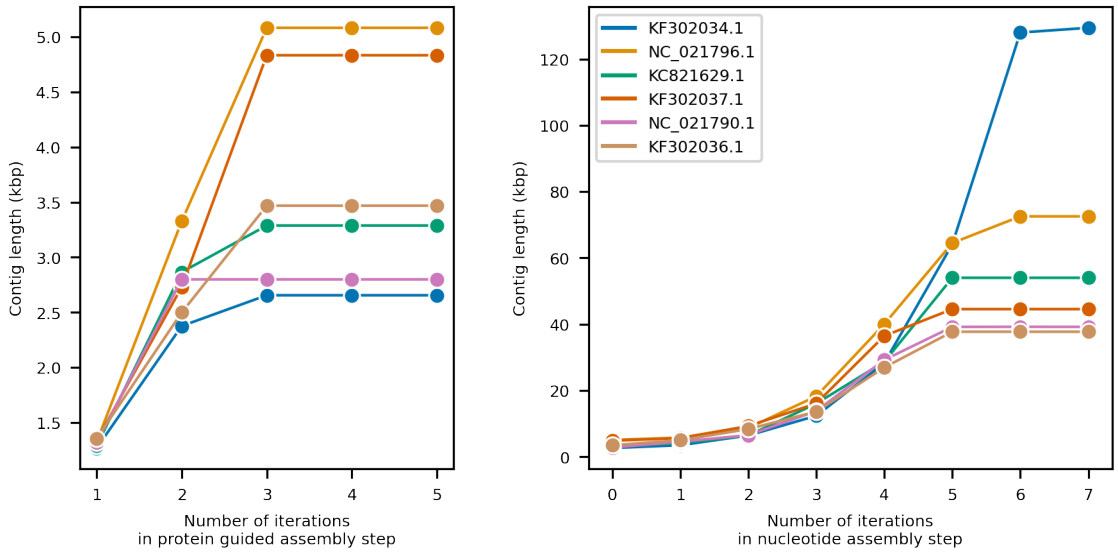


Figure 2.11.: Size of the largest contig per reference genome during PenguiN's assembly process for the mock community dataset. On the left is the growth of the contigs during the protein guided assembly iterations with a clear convergence at 2.5–5 kbp depending on reference genome, indicating completely assembled protein and ORF sequences. Dots indicate the contig length after the respective iteration. The right panel shows the growth of the contigs after switching to the nucleotide assembly iterations. The markers at 0 in the right panel correspond to the last ones in the left panel, but shown with the y-axis scaled differently, as the final genome lengths are much larger than the protein coding sequences. Towards seven iterations, all contig lengths have converged towards the full genome length. The largest contig per reference is determined by running MMseqs2 to search the reference genomes through the set of contigs after each iteration and selecting those with the largest alignment for each reference genome.

Extract viral contigs. In total, PenguiN assembled 42 contigs with a minimum length of 1000 bp on the mock dataset. From these, 30 were identified as putative viral contigs using VirSorter2 [Guo et al., 2021]. PenguiN is designed to deal with the challenges of viral diversities in genome assembly from metagenomic samples in mind. However, it is not limited to viral genomes only (no filter step is included), and therefore it also assembles all non-viral reads as well. Further investigation of the 12 contigs that were identified as non-viral showed that they were all much smaller (average length of ~ 2 kbp compared to ~ 19 kbp) and had high confident matches ($> 99\%$ sequence identity) to cellular organisms when searching against the NCBI nucleotide database (version from 2022-01-31) with MMseqs2. This indicates that they probably stem from contamination in the dataset and are not wrongly assembled contigs.

Next, PenguiN's assembly quality was evaluated against the reference genomes of the six *Caudovirales* present in this dataset, which were obtained from the NCBI RefSeq [O'Leary et al., 2016] database (accession numbers listed in Table 2.3), using MetaQUAST [Mikheenko et al., 2016a]. For all six genomes, MetaQUAST reports that the respective contig with the largest alignment covers 99.8 to 100% of the full reference sequence (see Fig. A.4 in the appendix for

the MetaQUAST report). The additional viral contigs that map to references were all much shorter and represent either (1) unidentified short duplicates or (2) deviations from the reference genomes that can be traced back to individual reads. These could either be biological, or artifacts due to the amplification or sequencing process. However, the exact origin of these reads could not be determined further. In total, 99.84% of the reads could be mapped back to the assembly according to the MetaQUAST analysis.

The accuracy of PenguiN's assembly was assessed using the number of mismatches and indels to the RefSeq genomes. Per 100 kbp, MetaQUAST reports 172.31 mismatches and 10.85 indels, corresponding to a total error rate of approximately 0.18%. As this is close to the expected Illumina sequencing error rate [Stoler and Nekrutenko, 2021] and PenguiN does not have its own logic to reduce sequencing errors, these errors can be attributed to sequencing errors, which are incorporated into the contigs during the assembly process. Pre-processing the reads with my own sequencing error correction tool CoCo (described in chapter 3), before running PenguiN's assembly, could improve the accuracy, reducing the error rate within the contigs to 0.013% (8.75 mismatches per 100 kbp, 3.98 indels per 100 kbp).

2.4.4. Assembly of ssRNA phages from real metatranscriptomic samples from activated sludge and aquatic environments

On real datasets, no ground truth is available, but the quantity of the assembled genomes can be compared among assemblers and the assembled genome's architecture can be more closely examined and evaluated using previously known characteristics.

To evaluate PenguiN's performance on large scale environmental data, I chose 82 public available metatranscriptomic samples from activated sludge and aquatic environments, which were recently used in a study to detect ssRNA phages from these two environments [Callanan et al., 2020]. In this study, Callanan and colleagues could expand the number of known ssRNA phage genomes using a Hidden Markov Model (HMM) approach to detect all sequences encoding a maturation protein (MP), a coat protein (CP) and an RNA-dependent RNA polymerase (RdRp) in contigs assembled with rnaSPAdes (version 3.12.0). Here, I used their model to identify sequences encoding the three ssRNA phage core proteins from PenguiN's assemblies and compared them to those of the Callanan study as well as other assemblers.

Dataset characteristic and preprocessing. In total, the dataset contained 16 activated sludge samples from Austria, 42 activated sludge samples from Illinois, 12 activated sludge samples from Japan, 4 freshwater aquatic samples from Lake Mendota (Wisconsin), 4 aquatic samples from the Mississippi River (Louisiana), and 4 freshwater aquatic samples from Singapore. I downloaded the samples from the NCBI Sequence Read Archive (SRA) database according to the accession numbers provided by [Callanan et al., 2020] (see Table A.1 in the appendix) and pre-processed them analogues to the Callanan et al. study using Cutadapt (version 3.3) [Martin, 2011] to remove Illumina adapters and Trimmomatic (version 0.39) [Bolger et al., 2014]

for quality trimming, but with updated versions. Thereby, all reads shorter than 100 bp were discarded.

Assembly. Afterwards, the remaining paired-end reads were assembled sample by sample with PenguiN, Megahit, metaSPAdes, metaviralSPAdes, rnaSPAdes, rnaviralSPAdes, Haploflow and VICUNA. Additionally, I also ran the combined pipeline of sequencing error correction using CoCo (described in chapter 3) and subsequent assembly using PenguiN. Notably, it was not possible to run SAVAGE (runtime >10 days per sample) and IVA on this dataset. Further, Haploflow and VICUNA did not finish for all samples within 10 days each, so only 76 samples could be examined for Haploflow assemblies (five "out of time" errors, one "out of memory" error) and only 49 for VICUNA assemblies (see tables A.2 to A.4). To be consistent with the analysis done by Callanan et al., only contigs with a minimum length of 500 bp were considered in the subsequent analysis.

2.4.4.1. Detecting ssRNA phage sequences in the assemblies

After assembling the reads with the different tools, I used the HMMs published by [Callanan et al., 2020] to detect ssRNA phage sequences. As their pipeline itself was not available, I rebuilt it from the description in the paper and their supplementary material. Thereby, I used mainly the same tools and parameters if specified. The whole workflow is depicted and explained in more detail in Fig. 2.12. In summary, non-redundant contigs (≥ 750 bp) encoding ssRNA phage core proteins (MP, CP, RdRp) are detected and filtered in three steps: contigs encoding at least two protein hits (partial genomes), contigs encoding three protein hits (near-complete genomes) and contigs encoding three protein hits without proteins that are prematurely terminated by the edge of a contig (complete genomes). Terms are used according to Callanan's analysis.

Using this pipeline, I identified 20 725 non-redundant contigs that encode ssRNA phage proteins in the PenguiN assemblies pooled from all samples. Of these, 13 999 contigs had a minimum length of 750 bp and included at least one core gene, 6469 included two core genes, 3501 included three core genes. Of these, 1954 were predicted to include the three core genes without premature termination by the edge of a contig (full-length genes), meaning an increase of 92.5% (939) compared to the 1015 of complete phages reported in the Callanan et al. study [Callanan et al., 2020].

Further investigation of the complete phage genomes showed very high similarity between some phages for both PenguiN and the complete phages available from the Callanan et al. study. Clustering with very strict settings (sequence identity $\geq$ 99.5%; coverage $\geq$ 99%) using the MMseqs2 cluster module [Steinegger and Söding, 2017] reduced the numbers to 1486 (76%) for PenguiN and 468 (47%) for the Callanan et al. phages. Clustering with less strict settings (sequence identity $\geq$ 97%; coverage $\geq$ 99%) even reduced this to 814 and 377 sequences. Judging if this similarity is due to redundancy or just to very similar strains is beyond the scope of this work. Nevertheless, it shows that PenguiN provides a 2 to 3-fold increase in the number of

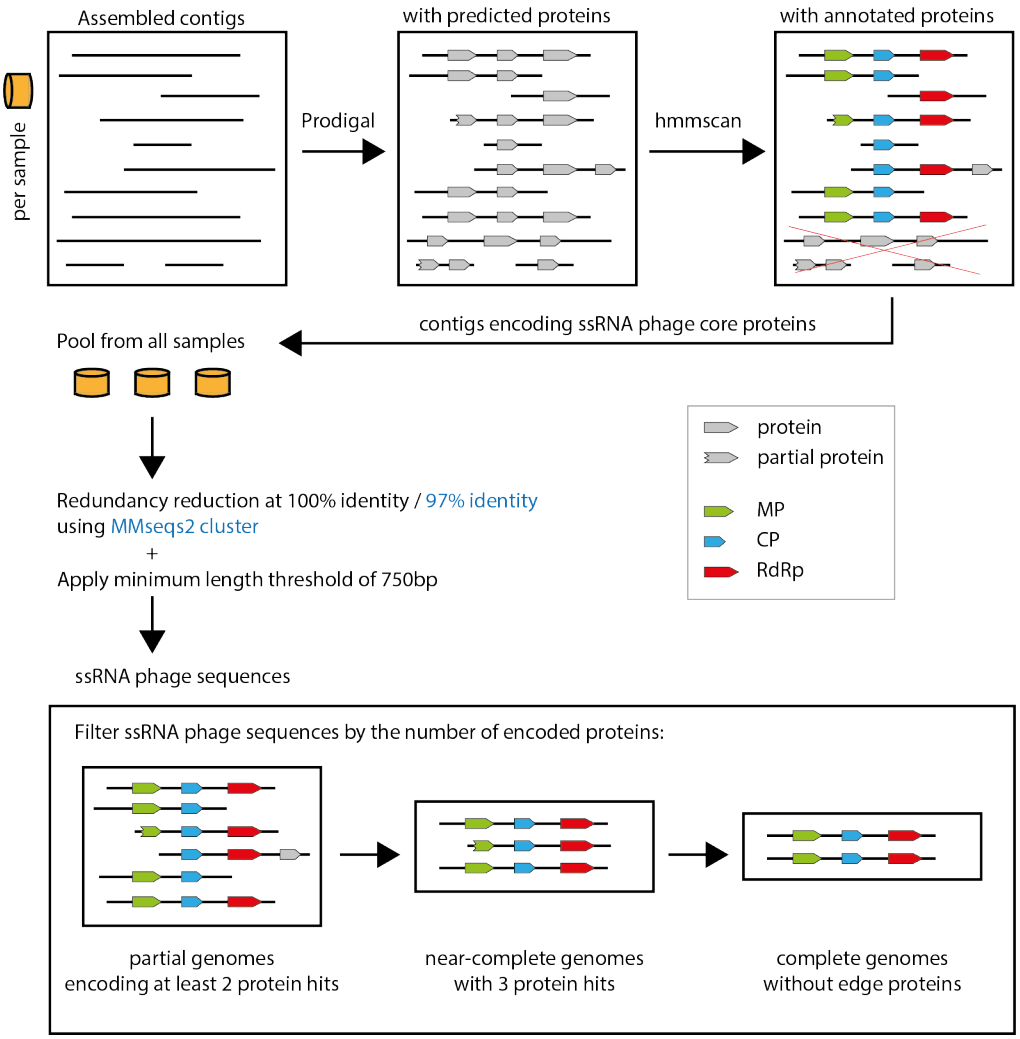


Figure 2.12.: Re-implementation of the ssRNA phage detection workflow based on the description of [Callanan et al., 2020]. Deviations from the original pipeline are colored in blue text. After assembly for each sample, proteins are predicted using Prodigal (version 2.6.3) [Hyatt et al., 2010] with options -p meta and -n. Predicted proteins are then searched against sequence profiles of the HMM 5-MC model generated by [Callanan et al., 2020] using hmmscan (version 3.2.1) [http://hmmer.org/]. Proteins with a hmmscan score of 30 or greater are annotated as the respective core protein. Then, all contigs encoding at least one ssRNA phage core protein (MP, CP, RdRp) are considered in the downstream analysis and pooled across all samples. Subsequently, redundancy is either reduced at 100% sequence identity according to the original description or at 97% using MMseqs2 cluster. Afterwards, non-redundant contigs of sufficient length (≥ 750 bp) including at least one core gene are reported as ssRNA phage sequences. Finally, contigs are filtered into different sets depending on the number of core proteins encoded within the contig: contigs encoding at least two protein hits (partial genomes), encoding three protein hits (near-complete genomes) and those encoding three full-length (no partial) protein hits (complete genomes).

complete phage genomes recovered compared to the Callanan et al. study, independent of the exact redundancy cutoff used.

To avoid giving too much weight to possibly redundant genomes, I used the more aggressive clustering/redundancy reduction settings (sequence identity $\geq$ 97%; coverage $\geq$ 99%) for the

(a)

	contigs with core protein hits	contigs non-redundant at 97% identity	contigs non-redundant at 97% identity and ≥ 750 bp
PenguiN	21 110	13 548	8869
PenguiN with CoCo	20 556	13 017	8609
Megahit	11 809	5895	3259
metaSPAdes	12 015	6189	3196
metaviralSPAdes	1	1	1
rnaSPAdes	10 699	4720	2680
rnaviralSPAdes	10 154	4545	2405
VICUNA	2319	957	623
Haploflow	5411	2063	1251

(b)

	contigs encoding at least two core proteins (partial genomes)	contigs encoding three core proteins (near-complete genomes)	contigs encoding three full-length core proteins (complete genomes)
PenguiN	3397	1472	**793**
PenguiN with CoCo	3371	1538	**833**
Megahit	1547	819	450
metaSPAdes	1477	770	417
metaviralSPAdes	0	0	0
rnaSPAdes	1381	821	453
rnaviralSPAdes	1153	613	354
VICUNA	356	225	134
Haploflow	584	302	122

Table 2.4.: Number of contigs in each class of the ssRNA phage detection pipeline applied to 82 metatranscriptomic samples from activated sludge and aquatic environments. Results are pooled across all 82 samples, however for VICUNA and Haploflow only 76 and 49 samples produced results. The results reported here are not directly comparable with those from Callanan's analysis [Callanan et al., 2020] due to the more aggressive redundancy reduction performed here. In their study, all non-redundant contigs at 100% sequence identity were considered, whereas here clustering at 97% identity was performed.

contigs of interest when comparing PenguiN's performance with those of the other assemblers (deviation from the pipeline is marked in blue in Fig. 2.12). The results are summarized in Table 2.4.

In all categories, PenguiN yielded the most sequences. PenguiN assembled up to 75-90% (343-376) more complete phages than the state-of-the-art tools Megahit, metaSPAdes, and rnaSPAdes and achieved around 6-6.5 times more complete phages than the viral assemblers Haploflow and VICUNA, which both did not produce results on all samples. The metaviralSPAdes assembler only produced a single contig across all samples that encoded for a single phage protein, and was therefore not considered for further downstream analysis.

When CoCo was additionally used for error correction upstream of the PenguiN assembly, the number of near-complete and complete genomes could even be increased slightly, while the total number of ssRNA phage sequences was slightly lower. This indicates that more contigs could

be assembled into whole genomes, which is probably due to the fact that the correction of the sequencing errors makes it possible to find previously overlooked overlaps. However, the additional use of CoCo was added later in this analysis to further improve PenguiN's assemblies and is not part of the following analyses, so the remaining comparison of the assemblers is not biased due to the use of an external sequencing error correction tool.

2.4.4.2. Pairwise comparison of complete ssRNA phage genome sets

To compare the assemblers among each other, I considered the overlaps of the sets of complete phages recovered by each assembler. The pairwise comparison shows that the set assembled by PenguiN covers $\sim$64-93% of those from the other assemblers, whereas these cover only $\sim$34-36% or much less (Haploflow (7.3%) and VICUNA (9.7%)) of the PenguiN assembly (see Fig. 2.13). The pairwise comparison between the other assemblers instead shows a significantly more balanced ratio ($\sim$60-70% in both directions), except for Haploflow and VICUNA, which both produced very little in total. This shows, that PenguiN can cover as much of the other metagenomic and metatranscriptomic assemblers as those cover among each other, hinting at the correctness of PenguiN's assembly. At the same time, PenguiN assembled many additional genomes, which were missed by the other assemblers.

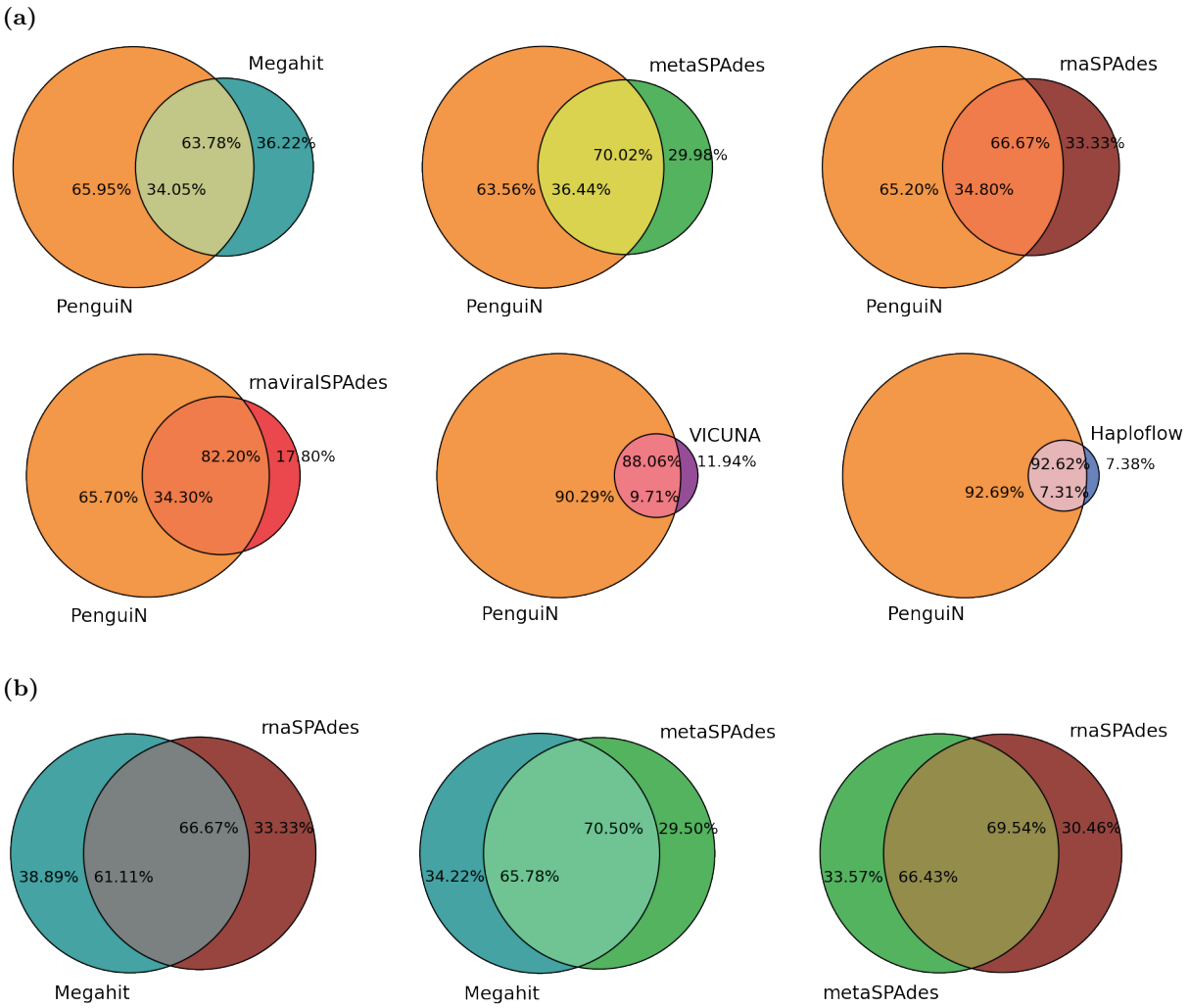


Figure 2.13.: Overlap of assemblies in terms of complete ssRNA phage genomes. (a) Overlap of complete phage genomes identified in the set assembled by PenguiN to those of six other assemblers (Megahit, metaSPAdes, rnaSPAdes, rnaviralSPAdes, VICUNA, and Haploflow). A phage genome in the assembly of tool A is counted for the overlap region when it is covered by an alignment to a complete phage in the assembly of tool B at a minimum sequence identity of 97% and minimum coverage cutoff of 99% (consistent with parameters used in the redundancy reduction). E.g. 34.05% of the complete phages in the PenguiN assembly are covered by alignments to complete phages in the Megahit assembly. Conversely, 63.78% of the phages in the Megahit assembly are covered by alignments with phages in the PenguiN assembly. VICUNA and Haploflow could cover significantly less of PenguiN's assembly than the other tools, which is a result of their low number of assembled complete ssRNA phage genomes (see also Table 2.4). (b) is the same as (a) but for the pairwise comparison among a selection of the other assemblers.

2.4.4.3. Further investigation of the assembled genomes

To investigate where the higher amount of complete phages in the PenguiN assemblies comes from and to assess the complete phage genomes in more detail, I performed further analyses, which I briefly summarize in the following.

Sample-wise analysis. First, I performed a sample-wise analysis using the same pipeline as described above to detect ssRNA phage sequences within the assemblies, but only clustered contigs within the same sample. This allows to evaluate the performance based on sample specific characteristics (e.g. sample location). The full results per sample are reported in the appendix (Table A.2, Table A.3, and Table A.4). Considering the number of complete phages per location showed that PenguiN can especially gain on the activated sludge samples from Japan and Illinois, whereas no complete phage was found among the 4 freshwater aquatic samples from Singapore (see Fig. 2.14). Notably, all assemblers detect significantly higher numbers of phages in the activated sludge samples (Austria, Illinois and Japan) than in the aquatic samples (Lake Mendota, Mississippi River, freshwater aquatic samples from Singapore). This was also previously observed in the Callanan et al. study for rnaSPAdes (version 3.12.0) and attributed to the fact that activated sludge might provide a better environment for the known hosts of ssRNA phages or technical differences (e.g. sequencing depth, microbiome complexity, ...) between the samples [Callanan et al., 2020]. In general, PenguiN followed the pattern as also seen for the other assemblers.

Diversity analysis. Next, I analyzed the diversity of the assembled complete ssRNA phage genomes. To this end, I used the full-length RdRp proteins extracted from each complete genome. The RdRp protein is the most conserved core protein and was used in the Callanan et al. study for potential taxonomic assignment on genera (AAI $\geq$80%) and species (AAI$\geq$50%) level [Callanan et al., 2020]. For each complete genome, I extracted the RdRp protein sequence from the protein translation file, which was generated with Prodigal [Hyatt et al., 2010] during the phage detection described above and assigned by hmmscan with a score $\geq$ 30. Thereby, only full-length RdRp proteins were considered. In cases where a complete genome encodes for more than one protein with RdRp protein hits, the one with the highest hmmscan score was chosen. So the number of extracted RdRp proteins was the same as the number of complete genomes for each assembler. The RdRp protein sets were then clustered at various levels using the MMseqs2 cluster module at sequence identity thresholds from 50% to 90% (in steps of 10%) and 90% to 100% (in steps of 1%) at a 99% minimum coverage of the shorter sequence. Thereby, the cluster runs were performed independently and not in a hierarchical fashion.

From these sets, I identified 771 unique RdRp sequences (X=100%) for PenguiN, 407 for Megahit, 397 for metaSPAdes, 396 for rnaSPAdes, 348 for rnaviralSPAdes, 133 for VICUNA and 122 for Haploflow, supporting the increase in sensitivity by PenguiN, which was already seen for the nucleotide sequences of the complete genomes (Table 2.4). The cluster trend is similar for

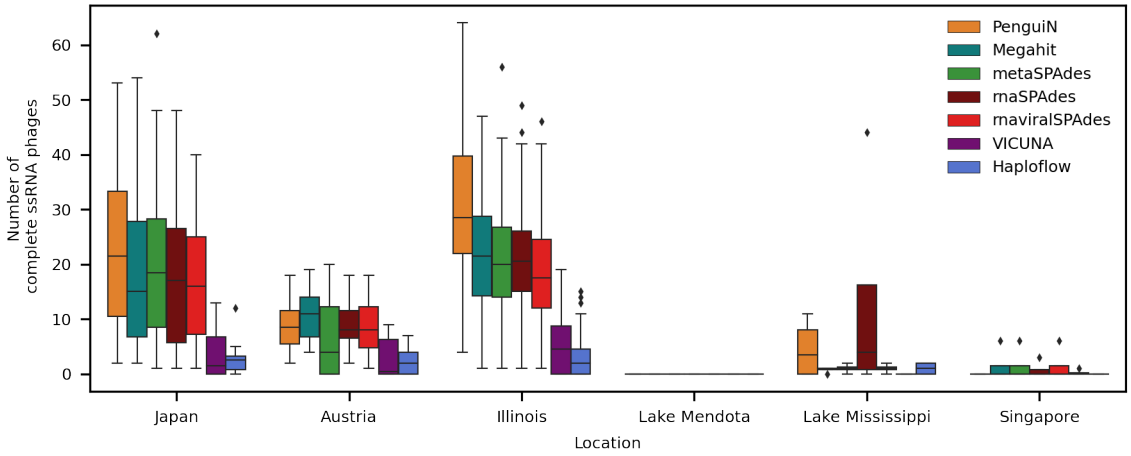


Figure 2.14.: Number of complete ssRNA phages identified in metatranscriptomic samples across different geographical locations. The 82 samples from the metatranscriptomic dataset are sampled from activated sludge from Austria (16), Illinois (42) and Japan (12) as well as aquatic environments from Lake Mendota/Wisconsin (4), Mississippi River/Louisiana (4) and freshwater aquatic from Singapore (4). All assemblers found significantly more complete ssRNA phages in the samples from activated sludge environments than in the aquatic environment samples. None of the assembler found a complete ssRNA phage genome in the samples from Lake Mendota.

Megahit and the three SPAdes variants (metaSPAdes, rnaSPAdes, rnaviralSPAdes): all four tools produce similar number of clusters, whereas PenguiN yields many more clusters at 90-100% sequence identity (see Fig. 2.15). However, at 50-80% sequence identity it is very close to the other assemblers. This suggests that PenguiN mainly gains at lower taxonomic levels as it can distinguish also very similar sequences, whereas those group to a very similar number of species (X=80%) or genera (X=50%) compared to those from the other assemblers. Haploflow and VICUNA yield a much lower number of clusters already at high sequence identity values due to the much smaller amount of complete phages assembled. However, the trend of their graphs also indicates that their assembled phages are rather dissimilar, as they mostly do not cluster together even at lower cluster values.

Finally, I also examined the characteristics of the ssRNA phages genomes assembled by PenguiN in terms of genome lengths and genome architecture and compared them to those from the other assemblers as well as to the previously known phages.

Genome lengths. For the 82 metatranscriptomic samples, all assemblers show a quite similar length distribution, with a median genome length of 3.8 kbp each (Fig. 2.16). The interquartile range of the genome lengths is thereby consistent with the lengths of ssRNA phage genomes revealed in previous studies (3.5–5 kbp) [Krishnamurthy et al., 2016]. However, almost all tools showed outliers with PenguiN (and PenguiN+Coco) producing the most: it produced 62 genomes >6 kbp (compared to 0-10 genomes), including 8 genomes >10 kbp. Nevertheless, more than 92% (731) of PenguiN's complete ssRNA phages genomes fall in the range of 2–6 kbp. Investigating the other sequences further, however, revealed that often the three core genes could be

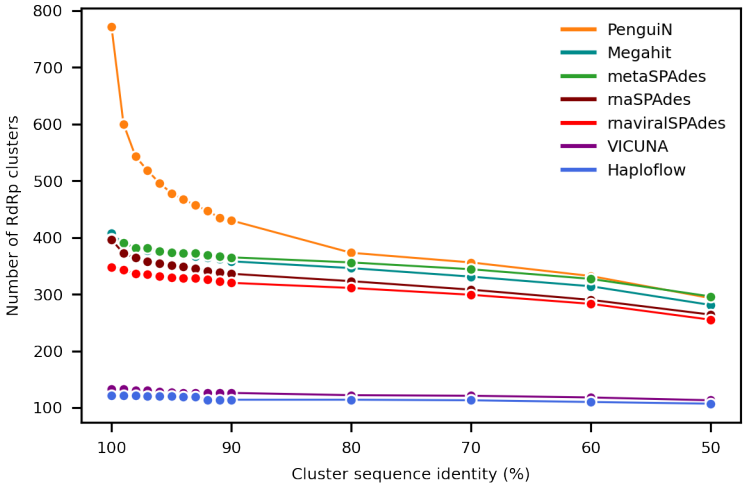


Figure 2.15.: Cluster analysis of the full-length RdRp proteins extracted from the complete ssRNA phage genomes. The clustering was performed using MMseqs2 cluster at sequence identity thresholds from 50% to 90% (in steps of 10%) and 90% to 100% (in steps of 1%) at a 99% minimum coverage of the shorter sequence.

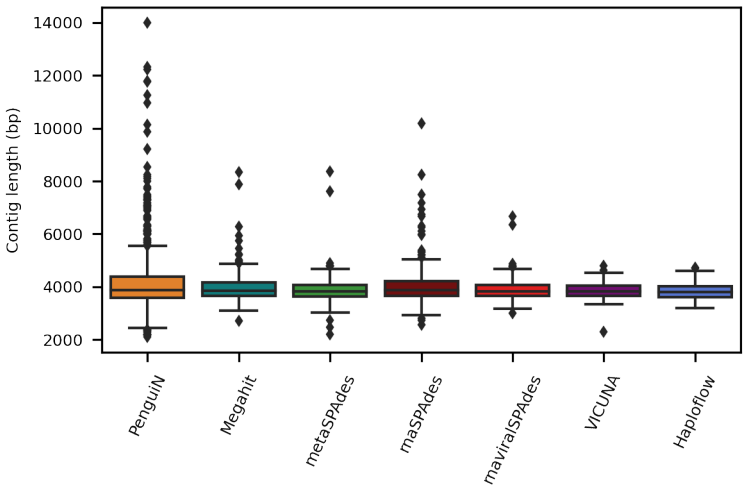


Figure 2.16.: Length distribution of the complete ssRNA phage genomes identified from the different assemblies.

still found within a range of 2.5–5 kbp, indicating the respective complete phage was probably assembled correctly but is embedded in a longer sequence. In the end, it could not be unequivocally clarified if these chimeras were true biological or misassembled artifacts. However, due to the minor fraction, the reasons were not investigated further.

Genome architecture. Another criterion, which was investigated, was the total number of full-length core proteins encoded by the sequences classified as complete phages. By definition, each of these sequences contained at least the three core proteins (MP, CP, RdRp) once. 754 (95.1%) instances of the PenguiN-assembled complete phages encode exactly three proteins that got full-length matches to the HMM sequence profiles of the three core proteins, 36 (4.5%) got an additional fourth and 3 (0.4%) had a fifth match. Further investigation of the additional proteins showed that, except for two cases, the additional protein followed the same kind, mostly an additional MP or additional RdRp hit directly following the native one, e.g. RdRp A directly following an RdRp A. The ssRNA phages are not known to have the core genes multiple times. However, the same could be observed for other assemblers as well. Previously, Callanan et al. has also recognized weak similarity between hypothetical genes following the RdRp and the native RdRp termini and theorized that these are either artifacts due to artificially introduced stop codons or can be attributed to the ability of phages to bypass stop codons during phage replication [Callanan et al., 2020].

I also considered the co-occurence of the core protein subtypes. The HMM 5-MC model released by [Callanan et al., 2020], which I used for the ssRNA phage detection (described in section 2.4.4.1), contained 3 MP, 3 CP and 2 RdRp protein subtypes. Except for one genome (contig:SRR7976310_60885), PenguiN maintained the same eight core-protein association profiles across the subtypes that were previously described in the Callanan et al. study (see Fig. 2.17). This is in line with our expectations, as the larger amount of complete ssRNA phages assembled by PenguiN is on a low taxonomic level, as shown in the diversity analysis, and therefore should not create new core protein association.

Additionally, PenguiN also maintained a specific order of the three core proteins. In 789 out of 793 cases, MP was encoded first, followed by CP and subsequently RdRp, which in turn is consistent with the specific order known for *Fiersviridae* (obsolete since 03/2021: *Leviviridae*) genomes [Tars, 2020]. This further supports the reliability of PenguiN's assembly and shows its ability to place the protein sequences into the correct genomic context.

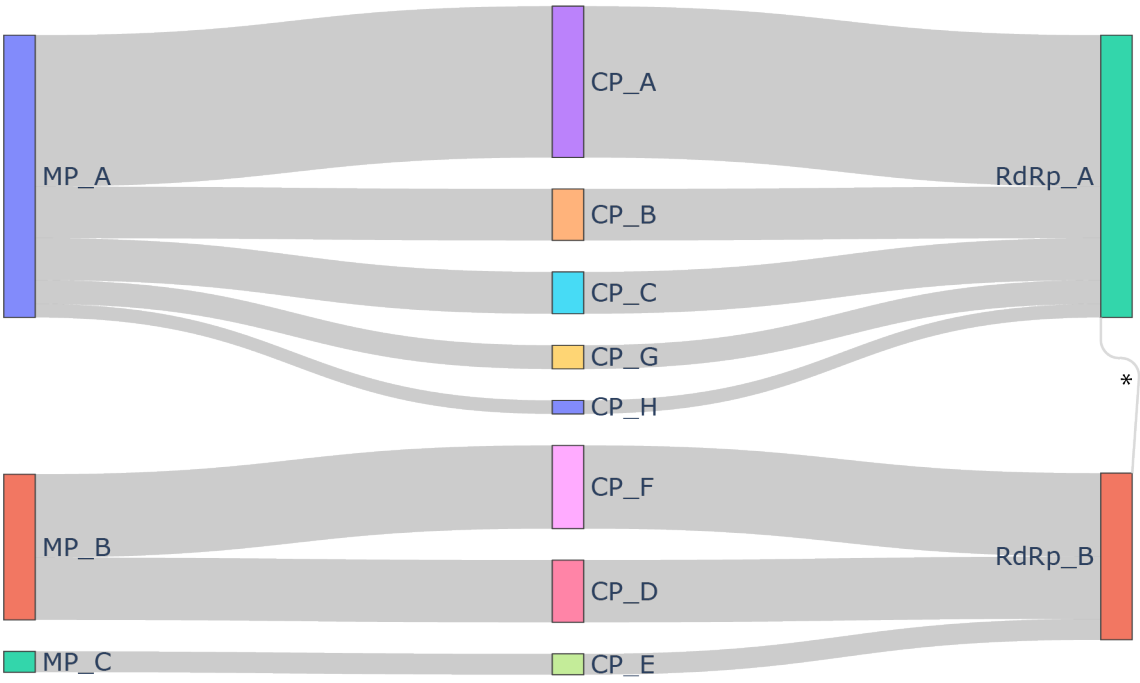


Figure 2.17.: Co-occurence profiles of the core proteins in complete ssRNA phage genomes assembled with PenguiN. The HMM profiles released by [Callanan et al., 2020] contained 3 MP, 8 CP and 2 RdRp protein clusters. Of the 48 theoretical combinations of these proteins on a complete genome, only 8 were observed in the PenguiN assembly, in accordance to the Callanan et al. study. There was only one deviation marked by an asterisks (*) where a contig (contig:SRR7976310_60885) contained an additional RdRp A following the RdRp B. However, the protein classified as RdRp A (hmmscan score 469.6) in this contig was also very similar to the RdRp cluster B (hmmscan score 240.0).

2.4.5. Computational time and memory usage

Furthermore, I assessed the runtime and memory usage of PenguiN in comparison to the eight other assemblers on the HRV in silico mixture, the three subsets (1-fold, 10-fold, 100-fold) of the 2550 HIV1 in silico mixture and the 82 samples of the metatranscriptomic dataset. All testing was completed using the GWDG HPC cluster (www.gwdg.de) on nodes with two Intel Xeon processors at 2.6 GHz totaling 16 cores and 128 GB RAM. Assemblies were thereby always performed on the local SSD (400 GB). Running time and peak memory footprint were measured using the /usr/bin/time command with the option -f "%e,%M". The runtime limit was set to 10 days on the HPC cluster system. For the metatranscriptomic dataset, assemblies were limited to 10 days per sample. All assembly tools were called with 16 CPU threads, except for Haploflow, which does not support multithreading.

Table 2.5 shows the total run time for all datasets. Due to the small size of the HRV dataset, the run time differences between the tools must be viewed with caution for this dataset. All tools were able to assemble the HRV dataset in very short time. Apart from that, Megahit, rnaSPAdes and rnaviralSPAdes had the shortest run times. They finished the three subsets of the HIV1 mixture in <0.5 min, <5 min, <10 min, respectively, and assembled the 82 samples of the metatranscriptomic dataset in less than 1.5 days. In contrast, metaSPAdes was by far the slowest tool from the SPAdes family used here, with more than 1 h for the 10-fold and more than 2 h for the 100-fold HIV1 dataset, even though it was run with the --only-assembly option. For the metatranscriptomic datasets, the runtime summed up to more than 27 days.

PenguiN was the fastest tool on the 1-fold HIV1 dataset and close to the best tools on the HRV, and the 10-fold HIV1 dataset. On the 100-fold HIV1 dataset it was 3.5-4.5 times slower than the fastest tools (Megahit, rnaSPAdes, rnaviralSPAdes), but 3.5 times faster than metaSPAdes and 47 times faster than IVA and VICUNA. For the 82 samples of the metatranscriptomic dataset, PenguiN took in total $\sim$7 days. This means 5-6 times slower than Megahit, rnaSPAdes and rnaviralSPAdes, but 4 times faster than metaSPAdes and much faster than VICUNA.

Dataset / Tool	HRV in silico mixture	HIV1 in silico mixture 1x	10x	100x	Metatranscriptomic dataset
PenguiN	00:08	00:21	00:04:19	00:36:55	166:17:23
Megahit	00:03	00:33	00:03:14	00:08:36	28:39:21
metaSPAdes	00:19	02:14	01:02:34	02:12:02	668:47:30
metaviralSPAdes	00:23	-	00:04:03	00:13:39	492:23:00
rnaSPAdes	00:08	00:27	00:02:14	00:08:06	34:28:50
rnaviralSPAdes	00:11	00:37	00:03:32	00:10:08	31:31:44
SAVAGE	01:41	25:27	> 10 d	> 10 d	-
IVA	03:04	04:31	00:51:31	29:17:30	-
VICUNA	00:02	00:33	03:28:33	29:31:19	*4989:38:20
Haploflow	00:02	00:47	-	> 10 d	*568:57:30

Table 2.5.: Comparision of runtimes for all datasets used in the assembly benchmark. Values are given in [hh]:mm:ss; values marked with * are underestimated because the tools did not finish on all samples.

Dataset / Tool	HRV in silico mixture	HIV1 in silico mixture 1x	10x	100x	Metatranscriptomic dataset
PenguiN	0.10	0.38	2.74	15.13	91.04
Megahit	0.25	0.27	0.29	1.81	10.96
metaSPAdes	5.11	5.15	22.16	46.97	116.66
metaviralSPAdes	5.11	-	5.67	11.59	41.52
rnaSPAdes	5.11	5.21	5.49	11.55	24.58
rnaviralSPAdes	5.11	5.21	5.47	11.55	17.64
SAVAGE	1.28	10.94	-	-	-
IVA	1.25	1.24	1.24	1.25	-
VICUNA	0.06	0.35	3.23	31.03	116.38
Haploflow	0.01	1.06	-	-	*119.00

Table 2.6.: Comparision of Max RAM usage for all datasets used in the assembly benchmark. Values are given in GB. (*)Haploflow had one sample in the metatranscriptomic dataset where the 128 GB were not sufficient for the assembly, resulting in an "out-of-memory error".

Overall, SAVAGE, IVA and VICUNA had the longest runtimes. From these, VICUNA was the only one which produced results on some samples of the metatranscriptomic dataset in less than 10 days each (for 49 out of 82 samples), however, the runtimes over all 49 samples summed up to more than 207 days. SAVAGE was by far the slowest program. It took more than 25 min for the 1-fold HIV1 subset and more than 10 days runtime for the 10-fold and 100-fold HIV1 subsets, respectively. It was also not able to assemble a single sample of the metatranscriptomic datasets in less than 10 days and was therefore excluded from the analyses of these datasets.

The recently published strain-resolved de novo assembler Haploflow [Fritz et al., 2021] was close to the runtimes reported for PenguiN. However, it ran into problems for the 100-fold HIV1 dataset and 6 of the 82 metatranscriptomic samples, where it did not complete within 10 days or had an out of memory error.

Table 2.6 shows the maximal memory used during the assembly of all datasets. PenguiN requires 91 GB to assemble the metatranscriptomic dataset. This is more than Megahit, rnaSPAdes and rnaviralSPAdes but less than metaSPAdes, VICUNA and Haploflow and results therefore in a similar pattern to that of runtime. The same applies to the 100-fold HIV1 dataset, whereas PenguiN needs much less memory than the three SPAdes variants on the HRV and the 1- and 10-fold HIV1 dataset. Due to the similar space used by the three tools of the SPAdes family, the core algorithm seems to have a minimum memory requirement of ~ 5 GB. Also most notable is the very low memory usage of Megahit across all datasets. IVA had a constant memory usage across the datasets, but, as described in the previous sections, it only produced very little contigs.

PenguiN automatically adjusts to fully utilize all available system memory, if necessary. The main memory limitation in PenguiN is the k-mer matching step, which has to extract k-mers from all sequences in each iteration. However, it also has the option to process k-mers in chunks to stay under a specified memory limit (set by the option --split-memory-limit). This should

enable PenguiN to process the datasets used here with less memory, but would increase the runtime. Further, it also gives the possibility to process much larger samples.

2.5. Discussion and Outlook

The aim of the study outlined in this chapter was the development of a new de novo metagenomic assembler that can deal with the high diversity of viruses and therefore assembles much longer viral contigs up to whole viral genomes from metagenomic samples in comparison to existing assembly tools. In the following, I discuss the results on the benchmark datasets as well as limitations of the assembler, possible improvements to the tool and two further applications.

As described in the introduction (see chapter 1), de novo assembly from metagenomic samples is a crucial task in virus research. The accurate reconstruction of the genomic sequences allows for the in vivo identification of newly emerging or previously unidentified viruses [Callanan et al., 2020; Gregory et al., 2020, 2019] and has implications for a deeper understanding of viral diversity, viral pathogenicity and virus-host interactions [Deng et al., 2021; Roux et al., 2015b]. These, in turn, can help improve environmental models and generate ecological hypotheses [Gregory et al., 2019; Roux et al., 2015b], as well as strongly influence the development of phage-based therapeutics approaches [Mirzaei et al., 2021].

Here, I provided a new approach that makes use of the enormous selective pressure operating on small viral genomes that limits most single nucleotide mutations between very closely related genomes to synonymous or conservative substitutions or non-coding regions (increased sensitivity) and at the same time makes use of the quickly diverging amino acid sequences to distinguish even strains, if the genome coverage is sufficient (increased specificity). The new method first assembles nucleotide sequences to full open reading frames simultaneously to the assembly of six-frame translated reads to protein sequences. Subsequently, it links the resulting ORFs from the first stage with nucleotide reads to bridge intergenic regions, resulting in longer contigs and ideally whole viral genomes. The resulting software tool PenguiN (protein-guided nucleotide assembler) adopts thereby the greedy iterative assembly strategy with an all-against-all overlap computation of the protein-level assembler Plass [Steinegger et al., 2019] for the two stages. The use of full-read overlaps enables PenguiN to be very sensitive to co-occurring changes in the sequences. With this, it overcomes the typical sensitivity-specifity trade-off of the de Bruijn graph assemblers that can only see co-occurences within one k-mer. The main contribution of this work was the extension of Plass to a full metagenomic nucleotide assembler. Due to its two level assembly approach, PenguiN assembles nucleotide contigs and ideally whole (viral) genomes, whereas Plass aims to assemble only proteins.

2.5.1. Add-ons to Plass

The main differences to Plass on the algorithm level is the simultaneous assembly of the underlying nucleotide sequences during the protein assembly and the subsequent nucleotide only assembly stage, which both required extensions and adaptations for nucleotide-specific considerations (e.g. reverse complement) as well as adaptations for longer sequences (e.g. scaling of the number of k-mers extracted per sequence) (see section 2.2.2). Further, I introduced a new extension choice strategy in PenguiN to deal with the difficulty of scoring overlaps with varying lengths (see section 2.2.2.4). Instead of using the non-overlapping residues of the alignment with the highest sequence identity value for the extension as done in Plass, the probability that an overlap is better than all others is calculated based on a Bayesian model. Beyond that, I added the possibility to detect circular contigs in PenguiN through a heuristic similarity approach (see section 2.2.2.5) and adapted the Linclust [Steinegger and Söding, 2018] algorithm, which I integrated in the PenguiN workflow for the redundancy reduction step, to cluster circular sequences as well (see section 2.2.2.6). Further smaller modifications introduced in PenguiN are the new end-to-end scoring mode, multiple extensions within the same iteration and the correct handling of asterisks, which mark the ends of protein sequences. Due to the different use-case, I also decided for much stronger default sequence identity cutoffs for the overlaps than used in Plass.

2.5.2. Performance

For benchmarking experiments and performance analysis, I considered several datasets. I tested PenguiN on two simulated datasets (including subsets varying in coverage), one in vitro mixture and one real metatranscriptomic dataset. At the time of the conception of the project, no metagenomic assembler exists, which could deal with the above-mentioned viral specific challenges sufficiently. There were mainly two types of assemblers (1) (general) metagenomic assemblers, which mainly employ the de Bruijn graph approach, and (2) viral specific assemblers including haplotype assemblers and consensus assemblers, which mainly utilize the greedy or the OLC assembly approach (see section 1.4). I compared the assemblies obtained using PenguiN with assemblers from both groups: SAVAGE [Baaijens et al., 2017], IVA [Hunt et al., 2015] and VICUNA [Yang et al., 2012] as they performed well on patient samples, and Megahit [Li et al., 2015] and metaSPAdes [Nurk et al., 2017] as the state-of-the-art metagenomic assemblers. Especially, the latter two were also recommended by previous benchmark studies on viral metagenomic data [Roux et al., 2017; Sutton et al., 2019]. I also integrated the metatranscriptomic assembler rnaSPAdes [Bushmanova et al., 2019] into my benchmark, as it was recently utilized for the assembly of RNA viruses from metatranscriptomic data in several studies [Callanan et al., 2020; Yan et al., 2021]. Later on, the three assembly tools metaviralSPAdes [Antipov et al., 2020], rnaviralSPAdes [Meleshko et al., 2022], and Haploflow [Fritz et al., 2021], which were published during the work on this project, showing the high interest in the research field of viral metagenomic, were added to the benchmarks as well. However, it should be mentioned

here, that the design of the benchmark was thereby probably not ideal for metaviralSPAdes, as the benchmark datasets contained RNA viruses. Assemblies were assessed based on common quality metrics using MetaQUAST [Mikheenko et al., 2016a] and calculation of sensitivity and precision values for the simulated datasets. On the real metatranscriptomic dataset, a ssRNA phage-specific pipeline was used for quality assessment.

I showed that PenguiN assembles long viral contigs up to whole viral genomes in high quality and is less influenced by the similarity of closely related genomes than other assemblers. The strain-resolved assemblers SAVAGE and Haploflow could still resolve closely related genomes from the HRV dataset containing only three genomes. However, they consistently suffered from the highly diverse strain mixture of HIV1 genomes, containing both very closely related genomes as well as more diverged genomes (ANI < 80). Further, they could not deal with the complexity and size of the metatranscriptomic samples. In total, they either produced much less than the others, resulted in small and fragmented contigs or did not finish in reasonable time (> 10 days). Also, IVA and VICUNA performed very poorly, obtaining only very low genome fraction with a high number of misassemblies on the simulated data. This is consistent with observations from previous assembly comparison studies [Deng et al., 2021; Sutton et al., 2019; Vázquez-Castellanos et al., 2014] and shows the limitation when applying those assemblers to highly complex metagenomic samples containing multiple strains and species. This may not be surprising, since both were designed for consensus assemblies from a viral population [Hunt et al., 2015; Yang et al., 2012].

In contrast, the metagenomic assemblers could deal with the size and complexity of the simulated data in principle better than the previous mentioned assemblers. However, due to the high viral diversity, especially in the HIV1 dataset (see section 2.4.2), they resulted in fragmented assemblies and a loss of variation. High genome fraction could only be reached when using multiple contigs together and aligning them at weak sequence identity thresholds ($\sim 90\%$) to the reference genomes. This hints for a consensus assembly (or chimeric contigs) and shows a lack of strain variation in the assemblies of the compared metagenomic assemblers. This observation is also in line with previous studies [Roux et al., 2017; Sczyrba et al., 2017; Sutton et al., 2019] and results probably from the limited possibilities to recover variation from de Bruijn graphs, as graph simplifications are necessary to traverse the highly branching graphs efficiently. As a further complication, the graph does only directly encode co-occurring mutations within the length of one k-mer, which hinders the resolution of closely related genomes. Notable, metaviralSPAdes performed consistently poorly on all datasets.

All tools were consistently outperformed by PenguiN, which could highly benefit from its full-read overlap approach, in terms of sensitivity while keeping a similar precision. On the HRV dataset, PenguiN assembled all three genomes within a single contig each. On the HIV dataset, PenguiN assembled by far the longest contigs and the most complete genomes within single contigs on all three tested coverage values and reached the highest per-nucleotide sensitivity independent of the considered accuracy cutoff (90-99%). For example, I obtained a 3 to 11-fold increase in per-nucleotide sensitivity compared to Megahit at similar or even higher per-

nucleotide precision. Notably, PenguiN achieved by far the highest sensitivity when contigs were aligned to the genomes at 99% sequence identity, whereas sensitivity massively decreased for all other compared metagenomic assemblers (see Fig. 2.10a). In contrast to the above-mentioned limitation of the other metagenomic assemblers resulting from the de Bruijn graph structure, PenguiN instead can utilize the co-occurrence of mutations within one read length due to its all-against-all overlap computation within its greedy iterative approach. Similar to overlap-graph based assemblers like SAVAGE [Baaijens et al., 2017], it can therefore distinguish closely related genomes, but without the need for the memory expensive overlap graph structure [Li et al., 2012]. Thereby, the protein level assembly is a crucial part in PenguiN to reduce the problem's complexity. The sequences are much shorter and therefore less expensive to align. If the sequences do not match at the amino acid level, they will not be considered at the nucleotide level in stage I. Further, it allows for the use of "longer" and therefore more specific k-mers (k=14 on amino acid level corresponds approximately to a spaced k-mer of length 42 on the nucleotide level). At the same time, re-evaluation on nucleotide level, performed if an overlap on amino acid level is good enough, further ensures that precision is not lost on nucleotide level. Altogether, this allows to distinguish between very closely related genomes (ANI $\sim$ 97%), if the coverage is sufficient, and contributes to the out-standing high sensitivity at high accuracy cutoffs (see Fig. 2.10).

Moreover, I showed that PenguiN was the only tool that could reconstruct genomes from the complex HIV1 strain mixture by more than 90% on the 1x coverage set, where most likely the similarity of amino acid sequences of very closely related genomes can especially help to increase sensitivity. This is especially notable, since difficulties with low coverage or low abundant virus genomes is a known problem for de novo assemblers [Deng et al., 2021; García-López et al., 2015; Roux et al., 2017; Sutton et al., 2019].

Besides the benchmark on the simulated data, I showed, with the help of a lab generated mock community from a previous study [Warwick-Dugdale et al., 2019], that PenguiN also performs well on real sequencing data. With small adjustments of the default parameter settings (increased number of iterations), PenguiN can even assemble the very long dsDNA phages (38.2–129.4 kbp) from the mock community within one contig with high accuracy (99.82%). The missing accuracy is mainly caused by sequencing errors and can be reduced further by using a sequencing error correction tool such as CoCo (described in chapter 3). The additional shorter contigs, which were obtained as well, can hint at misassembled regions (as they did not cluster together with the longer contigs during the redundancy reduction), undetected redundancy or biological/technical deviations from the reference genomes.

With the application on the metatranscriptomic dataset from 82 aquatic and activated sludge samples, used in a previous study for ssRNA phage discovery [Callanan et al., 2020], I showed the enormous impact of PenguiN to resolve previously uncharacterized viral diversity from metagenomic short read sequencing data. PenguiN assembled 75-90% (343-376) more complete ssRNA phage genomes than the state-of-the-art tools Megahit, metaSPAdes and rnaSPAdes and many times more than VICUNA and Haploflow. The reliability of the tremendous expansion in the

quantity was thereby carefully checked in several downstream analyses (see section 2.4.4.3). Besides a few cases (< 8%), which showed much too long genome lengths and might be a result of misassemblies, there seems to be no reason for doubt on the correctness of the phage genomes recovered by PenguiN. They lie in the expected genome length range, with the correct genome architecture (number of core proteins and order) and do not even show any deviations from the protein co-occurrence pattern discovered in a previous study [Callanan et al., 2020]. Moreover, a diversity analysis on the RdRp protein suggests that PenguiN achieved approximately the same number of potentially different species and genera, but especially outperforms the other assemblers for more closely related genomes (80-100% sequence identity on the RdRp protein) and achieved many more complete phage genomes in total. This confirms the results from the simulated data and might have strong impact on our understanding of phage biology and diversity in the future.

Thereby, PenguiN is sufficiently fast, and its memory consumption is reasonable. For the assembly of the 82 metatranscriptomic samples it took a combined runtime of ~ 7 days in total, meaning an average runtime of ~ 2 hours per sample, at a maximal memory peak of ~ 91 GB. In practice, the waiting time was much shorter, since multiple assembly jobs could be run in parallel on the cluster system. On the smaller simulated datasets (total size of 7.8 MB to 4.6 GB) PenguiN's runtime and memory requirement was among the best of the tools tested. For the metatranscriptomic dataset its runtime is considerably slower than Megahit and rna(viral)SPAdes, but much faster than metaSPAdes, Haploflow, VICUNA, SAVAGE and IVA. Possible improvements of the run time of PenguiN are discussed in section 2.5.4. However, I think PenguiN's run time and its memory consumption is already reasonable, especially when taking the significantly higher sensitivity into account.

2.5.3. Shortcomings and limitations

The results provided in this work suggest that PenguiN outperforms current state-of-the art methods when considering the viral fraction of metagenomic samples. There are, however, limitations which I discuss in this section.

One of the first points to be noted is that PenguiN has no built-in error correction. Whereas de Bruijn graph assemblers usually employ the topology of de Bruijn graphs for direct or implicit error correction [Heydari et al., 2017], overlap-graph based or greedy iterative assemblers utilizing full-read overlaps usually do not have any sequencing error correction strategy. They can tolerate a small amount of sequencing errors in their alignment-based overlap computation and often exclude them through a consensus step at the end [Li et al., 2012]. PenguiN falls in line with the latter ones. It works on the input reads directly without any correction, allowing for a small number of sequencing errors as expected from today's highly accurate Illumina sequencing data (<<1% errors, mostly substitutions) [Stoler and Nekrutenko, 2021] through non-exact overlaps (sequence identity cutoff $\geq$99%) and the subsequent clustering step. Using all reads as they are has advantages for low frequency variants and low coverage regions, which might be lost

through a sequencing error correction procedure otherwise [Heydari et al., 2017; Yeom et al., 2019], but also has the disadvantage of missing connections, especially when the sequencing errors cumulate at the end of the reads [Schirmer et al., 2016; Stoler and Nekrutenko, 2021]. In this case, it may happen in PenguiN that no common k-mer can be found for overlapping reads or alignments describing overlaps would not full-fill the sequencing identity cutoffs. This is especially crucial at lower coverage, where these reads might be crucial to assemble longer contigs and not using them would result in fragmented assemblies. Further, independent of the coverage, reads containing cumulative sequencing errors at the opposite end to that which is used in the previous iteration can result in dead ends for the greedy iterative assembly strategy. Using a trimming tool such as Trimmomatic [Bolger et al., 2014] to discard parts of the reads with low per-base quality and applying a sequencing error correction tool before the application of PenguiN can therefore help to improve the assembly result further. One choice I tried here is the sequencing error correction module of CoCo, which I developed during my doctoral research as well, and also describe in the next chapter (see chapter 3). Further, it should be mentioned that the current PenguiN implementation cannot take insertion or deletion errors in the overlap region into account, as it only computes an ungapped alignment to score an overlap. This is only a very minor issue for Illumina's short sequencing reads, where a rate of approximately 10^{-6} indel errors per base is expected [Schirmer et al., 2016], but might be crucial when considering the application on long sequencing reads (see below).

Another issue that needs to be discussed is the slightly higher redundancy obtained in PenguiN's assemblies for the HIV1 datasets. To overcome the issue of dead ends in low coverage regions during the greedy iterative assembly strategy, PenguiN re-uses reads. More precisely, different contigs can be extended with the same read. However, this can result in the same genome or genomic region being built multiple times in parallel. If the coverage is high, this becomes even more extreme, as the same sequences can be built from different redundant reads, leading to highly redundant assemblies. Due to the choice of the hash-function to pick k-mers in the overlap phase and then always taking k-mers with lowest hash-values, the effect is minimized. In most cases, the same k-mers will be selected and redundant reads therefore assigned to the same center sequences, however it cannot be prevented completely. In other greedy assemblers such as PRICE, a similar behavior was observed and reduced by collapsing fully overlapping contigs in a subsequent meta-assembly step [Ruby et al., 2013]. To overcome this issue in PenguiN, I integrated the Linclust [Steinegger and Söding, 2018] algorithm and only output the cluster representatives as final contigs. This massively reduced the redundancy. However, Linclust's speed comes at the expense of some loss in sensitivity [Steinegger and Söding, 2018]. Therefore, some sequence matches could be missed, and a small amount of redundancy remains. In cases where redundancy is problematic (e.g. when comparing quantities of assembled genomes), I suggest using a more sensitive all-against-all clustering in a post-processing step after the assembly. In the benchmarks presented in this work, additional clustering was performed using the new nucleotide_clustering workflow of the MMseqs2 software suite, which combines Linclust with the more sensitive MMseqs2 all-against-all search for forward and reverse strand followed by a clustering step [Hauser et al., 2016; Steinegger and Söding, 2017]. It should also be mentioned

here, that redundancy was also observed for other assemblers previously [Deng et al., 2021]. Another possible workaround to reduce the redundancy for PenguiN, which I have not tried yet, would be to cluster the reads beforehand. Since reads can be re-used in PenguiN, this should not restrict the assembly process, but decrease redundancy and most likely also the run time.

Another, more notorious problem associated with the greedy approach is that overlaps are evaluated locally, which does not lead necessarily to the globally optimal sequence [Miller et al., 2010; Pop, 2009]. Therefore, single genome assemblers utilizing the greedy approach usually terminate the assembly process to avoid misassemblies if conflicting extension are found [Jeck et al., 2007; Warren et al., 2007]. However, due to the presence of multiple, closely related genomes in metagenomic samples, this would lead to highly fragmented assemblies. In PenguiN, I do not use such a strategy and always extend with the best sequence according to the Bayesian formulation described above. In general, this comes with the risk to join unrelated sequences (misassemblies) triggered by a common sequence. However, in the benchmarks performed in the scope of this work, only very few misassemblies were observed in the PenguiN assemblies. I think this is due to the characteristics of viral genomes and the two level assembly approach. Either two genomes only differ by very few mutations, from which most are silent or conservative mutations, then a contig combining them still represents both with high accuracy (consensus), or the mutations are so many due to the fast evolution of viral genomes that the read becomes "unique" and the right choice can be made. Assembling viral genomes with a greedy iterative assembly approach seems therefore a good choice, and worked previously also in other studies [Deng and Delwart, 2021; Hunt et al., 2015]. However, I speculate the chief limitation is when it comes to more complex genomes containing long repeats (longer than a read length) or highly conserved regions. Using PenguiN for such data comes with the risk of many chimeric connections and will probably require further optimization. Further, I want to point out here, that the aim of the project was not to resolve any single sequence variant and that this would also not be possible with the PenguiN's assembly strategy.

Finally, I want to discuss the limitation of PenguiN for long reads, the newest sequencing technologies. The two dominating approaches, single-molecule real-time (SMRT) sequencing [Eid et al., 2009] from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies' (ONT) nanopore sequencing [Manrao et al., 2012], currently produce very long reads of commonly 10–30 kbp [Amarasinghe et al., 2020] up to records of even a few megabasepairs [Payne et al., 2019]. The read length itself is not an issue as there is no limitation for the input read length in PenguiN and the number of k-mers extracted from each sequence is scaled with the sequence length. However, the issue with these data for PenguiN is highly connected with the first point mentioned in this section as they have very high error rates, ranging from 3-15%, of which most are indels [Zhang et al., 2020]. Assembling genomes from these sequencing technologies would therefore require much lower sequence identity cutoffs for the overlapping regions, as well as gapped alignment computation in PenguiN to take indels into account. However, the chief limitation would probably be frame shifting indel errors, which undermine the amino acid sequence guided assembly stage of PenguiN. Therefore, PenguiN is currently not capable to work with

these data until further technological developments decrease the rate of indel errors. In future research, long reads could be a great opportunity for viral research as they can cover whole viruses [Pinzone et al., 2019; Takeda et al., 2017] and therefore even circumvent the assembly step in a metagenomic sequence analysis [Mirzaei et al., 2021]. However, it remains questionable if the error rate can be reduced so drastically that the true viral diversity in a natural viral community can be revealed and that the nanograms of DNA/RNA usually obtained for the viral fraction of a metagenomic sample are sufficient for the long read technologies [Mirzaei et al., 2021].

2.5.4. Possible improvements / further work

PenguiN outputs the cluster representatives found by Linclust in the redundancy reduction step as the final master contigs. Thereby, the cluster representative is always the longest sequence within the cluster. However, to represent the group of sequences as good as possible, it could be worthwhile to validate the nucleotides of the representative sequences using the cluster member sequences and output a consensus sequence for each group. This would then automatically also reduce the chance of reporting sequencing errors within the final contigs. However, it should be noted that the calculation of a consensus can lead to a sequence from the cluster subsequently showing a greater distance to the final sequence. Whether the consensus is therefore the best choice remains to be tried out.

During my doctoral research, I have observed well-working default parameters. However, adjusting the parameter --num-iterations to other data might be confusing, as the extension for each sequence is hard to estimate. Therefore, I would like to integrate an additional stop criterion for the greedy iterative assembly strategy. If I keep track of the extended sequences in each iteration and only extend those in the next iteration, then a convergence can be reached when no further extension can be made.

Excluding the sequences, which were not extended, from the possible center sequences in subsequent iterations could also be a strategy to improve upon speed. A second opportunity to decrease run time is to collapse fully identical reads. Having the exact same read sequence multiple times in the input set is a sign of very high coverage and does not contribute any further information for PenguiN's assembly, but increases run time and memory usage. A third strategy to decrease run time would be to prevent the re-use of reads. However, as discussed above, re-using reads has several advantages for low coverage regions or for low abundant genomes, especially when the greedy strategy runs into dead ends. Therefore, I suggest checking the influence of the latter one cautiously and integrate it only optionally through a new parameter. The user can then choose between a more sensitive but slower assembly or a faster, but less sensitive.

For better user experience, I also plan to integrate the sequencing error correction with CoCo directly into the PenguiN workflow. The same is done for example with BayesHammer [Nikolenko et al., 2013] in metaSPAdes [Nurk et al., 2017]. However, integrating CoCo into PenguiN would

also provide further possibilities. One could for example utilize the k-mer count profiles of CoCo as additional information to evaluate overlaps between two sequences. I would expect that the position dependent k-mer counts of the suffix of a sequence to be similar to that of the prefix of the sequence to extend with if the suffix-prefix pair form a true overlap in the genome. Another, but related, possibility would be to integrate tetranucleotide frequencies in the overlap evaluation. It was shown that the tetramer composition along the genome is conserved well enough to distinguish contigs from different microbial genomes [Kang et al., 2015; Pride et al., 2003; Saeed et al., 2012]. Integrating this information could therefore be useful to avoid chimeric assemblies, especially for more complex microbial genomes.

A further possible extension for PenguiN is scaffolding. Due to the non-uniform sequencing coverage, gaps can occur that can not be closed during the assembly process, leading to fragmented contigs [Hunt et al., 2014]. One typical technique to get more contiguous draft genomes is to determine the order and orientation of contigs belonging to the same genome with the help of paired-read information [Huson et al., 2002; Kim et al., 2008]. Currently, PenguiN assembles only contigs, but could be complemented by a typical scaffolding step afterwards in which paired-end reads are mapped onto the contigs and then linked with a certain number of N's in between depending on a given or estimated insert size. However, based on the smaller sizes compared to microbial genomes, viral genomes are more frequently assembled in single contigs already.

Finally, PenguiN could be extended and adapted for other applications as well. Two examples that are currently considered are transcriptome assembly and the assembly of metagenomic ancient samples. They are briefly outlined in the next subsection.

2.5.5. Further applications

De novo transcriptome assembly. De novo transcriptome assembly aims to reconstruct the transcriptomes from short read RNA sequencing (RNA-seq) data without the need for a known genome sequence, allowing for the identification of transcripts and expressed genes at a certain time point [Hölzer and Marz, 2019; Martin and Wang, 2011; Raghavan et al., 2022]. The task is more related to metagenomic assembly than to genome assembly, as it shares major difficulties like the varying abundance due to the different expression levels of the transcripts [Conesa et al., 2016; Martin and Wang, 2011]. However, it also brings its own difficulties, such as transcript isoforms due to alternative splicing [Canzar et al., 2016; Ozsolak and Milos, 2011; Wang et al., 2008]. In the scope of his master thesis [Kraft, 2021], Louis Kraft evaluated the applicability of PenguiN for transcriptomic data under my supervision, and compared the obtained assemblies with the state-of-the-art transcriptome de novo assembler Trinity [Grabherr et al., 2011]. Even though PenguiN could not outperform Trinity, his results suggest that PenguiN in its native version performs almost as good as Trinity in terms of sensitivity, precision, and minimum fraction identity and revealed similar patterns in the BUSCO [Simão et al., 2015] scores. However, PenguiN produced much more redundancy and Trinity generated significant

longer contigs on average. Some algorithmic adaptations to further improve the assembly were considered, but could not make a significant difference on the transcriptomic data yet. Especially, it stays questionable if the misassemblies in transcriptomic data, triggered by common sequences, can be avoided in PenguiN's assembly strategy. However, to provide a reasonable competitor to Trinity in practice, PenguiN would also need to provide isoform information for the assembled transcripts allowing for the differential gene expression analysis.

Assemble ancient metagenomic samples. Another application of PenguiN that is currently considered is the assembly of environmental ancient DNA (aDNA) samples. Assembling ancient DNA is especially challenging due to the degeneration of the DNA sequences over time that results in short and damaged fragments. Here, an overlap-based assembler can have an advantage, as mismatches in the overlap region can be accounted for. Antonio Fernandez-Guerra's group at the Globe Institute / University of Copenhagen has recently started to utilize PenguiN in their pipeline for the assembly of ancient DNA samples. They have assembled data from different environments, ages, and read lengths, some published others unpublished. The samples are from permafrost [Wang et al., 2021], marine sediments, dental calculus, and chewing gum [Jensen et al., 2019]. The ages span from the oldest permafrost samples from Greenland which are 2M years old to the youngest which are 1000 years old. The shortest read lengths have been ∼28nt while the majority of them have a modal length ∼50nt. They assembled the data using Megahit and PenguiN (utilizing spaced k-mers and a lower sequence-identity threshold in the guided stage, -k nucl:22,aa:7, --spaced-kmer-pattern nucl:"",aa:110111011, --min-seq-id nucl:1.0,aa:0.9). Afterwards, they dereplicate the contigs with MMseqs2 and carefully check for chimeras (intra-genomic and inter-genomic). They observed that PenguiN shows higher performance in discovering the ancient fraction, whereas Megahit tends to recover what would be the "living fraction". Thereby, they see that both methods make mistakes, but these can be controlled within their pipeline. The results are not yet published, but a publication is currently in preparation.

2.6. Conclusion

Taken all together, my doctoral research showed that assembling viral genomes from metagenomic samples is challenging, but also demonstrated that the approach presented in this chapter can overcome the major drawbacks of both, (general) metagenomic and viral specific assemblers.

PenguiN assembles significantly longer viral contigs and more complete viral genomes from complex samples than existing assembly tools in a reasonable time. It overcomes the variation loss in metagenomic de Bruijn graph assemblers and deals with the size and complexity of metagenomic samples much better than overlap-based viral specific assemblers. Moreover, it is competitive to the existing metagenomic assembly tools for more distantly related viral genomes and outperforms them for more closely related genomes, distinguishing even very similar sequences. Thereby, it can also reconstruct genomes of low frequency. Its main advantage seems to be

the two level assembly approach combined with the greedy iterative assembly strategy and the all-against-all full-read overlap computation. Thereby, especially the protein-guided assembly stage allows for a simplification of the assembly problem itself: (1) genomes that already differ on amino acid level do not have to be compared on the longer and therefore more computationally expensive nucleotide sequences, and (2) at the same time, single nucleotide mutations in the coding regions, which do not change the amino acid sequence significantly (synonymous or conservative mutations), do not confuse the assembly process. Additionally, Penguin's full-read overlap based approach allows it to utilize all information encoded in a read. It thereby has an advantage over de Bruijn graph assemblers, which split reads into k-mers when constructing the graph. PenguiN can instead utilize the co-occurence of SNPs within one read-length to distinguish closely related variants. Consequently, PenguiN can assemble many more complete viral genomes than the tested de Bruijn graph assemblers, even on highly complex datasets of thousands of closely related viral genomes.

Thus, I hope PenguiN can help to recover previously uncharacterized viruses and viral diversity in a variety of environments in future studies and therefore help to expand our limited knowledge of viruses (especially phages) occurring in nature. Getting to know the genomes is thereby a key step on the way to understand the effect of the viral component in microbial communities and can help to improve upon environmental models and understanding virus-host interactions for a variety of ecosystem ranging from geochemical cycling to human health. Further, it has the potential to discover newly emerging viruses.

3. Sequencing error correction based on spaced k-mer count profiles

During the development of the PenguiN assembler, which I introduced in the previous chapter (see chapter 2), I realized that sequencing errors can sometimes hamper the assembler in finding overlaps between reads or read and contigs. This seems to be especially the case if a region is only sparsely covered, and the read overlaps are therefore small. Sequencing errors could then lead to overlaps being overlooked if no matching k-mers can be found, or ignored if they do not pass the sequence identity threshold. This becomes even more problematic, as sequencing errors accumulate at the 3' end of reads [Stoler and Nekrutenko, 2021]. In highly covered regions this is obviously less problematic as the read overlaps usually become long enough so that enough correct position can compensate for sequencing errors. However, I argue that also these can benefit from cleaner overlap as a stronger sequence identity threshold can be chosen. In addition, the quality of the final contigs can be improved if fewer errors are included. Currently, PenguiN does not have its own logic to avoid integrating sequencing errors. In this chapter, I propose CoCo (**Co**nsensus **Co**rrection), a new sequencing error correction tool, exploring the discontinuity in spaced k-mer frequencies along sequencing reads to identify and correct sequencing errors. Due to the work outlined in this chapter, CoCo can be applied pre-assembly to PenguiN and helps to assemble more contiguous contigs due to cleaner read overlaps and can help to reach better contig quality. CoCo is not yet published, but a manuscript is currently in preparation for the publication subsequent to this thesis. Thereby, parts of the following text and figures will be included.

In the following, I first introduce the algorithm and the underlying idea, then I outline the details of the implementation including the underlying data structure to efficiently store spaced k-mer counts. Afterwards, I show the results on two simulated datasets and illustrate how CoCo impacts PenguiN's assembly. Finally, I discuss the results in a broader context and give a perspective of the further development that remains to be done.

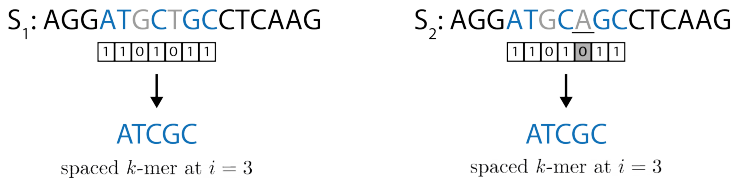
3.1. Algorithm

3.1.1. Terminology and Notation

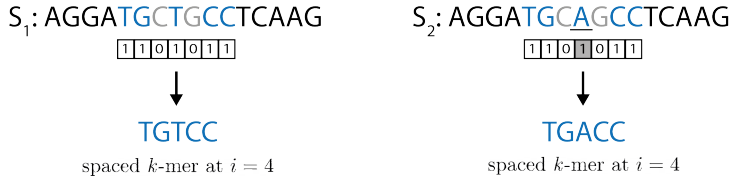
For a given sequence S of length $L = |S|$ over an alphabet $\mathcal{A}$, a substring $S[i]$ denotes the i-th element in S and $S[i..j]$ with $0 \leq i \leq j \leq L-1$ the contiguous substring of S from i to j. In particular, a substring of S of length k, denoted by $S[i..i+k-1]$ for $0 \leq i < L-k+1$, is termed k-mer. Analogously, a spaced k-mer is a non-contiguous substring of S of length k following a binary pattern P with $P \in \{0,1\}^{\ell}$ having exactly k many '1'. Thereby, a '1' in the pattern P is called an *informative* position, and a '0' is a *non-informative* position. We say that a spaced k-mer K_i with respect to some pattern P occurs in a sequence S at position i if $K_i = S[i+k-1]$ for all informative positions k of P. Whereas, the corresponding continuous substring of length ℓ, is called ℓ-mer. The number of *informative* positions in a pattern P is its weight, ℓ is its length.

As by definition, the spaced k-mer of S respectively to pattern P only contains the letters at the *informative* positions, two different substrings of length ℓ can lead to the same spaced k-mer. However, if the ℓ-mers differ at an informative position also the corresponding spaced k-mers differ.

For example, for two sequences S_1 and S_2 (shown below), which differ at position 7, and a spaced pattern $P = 1101011$ the spaced k-mer starting at position 3 in S_1 is the same as the spaced k-mer starting at position 3 in S_2.



Whereas, the spaced k-mer starting at position 4 is affected by the difference, as the difference is at an informative position.



This property I use in CoCo to identify sequencing errors (see section 3.1.2).

For a set of sequences, one can determine the frequency of a spaced k-mer over all sequences, and therefore obtain a count value $count(K)$ for each spaced k-mer K existing in the sequence set. For a k-mer K that does not occur in the set of sequences, I define $count(K) = 0$.

3.1.2. Spaced k-mer count profiles

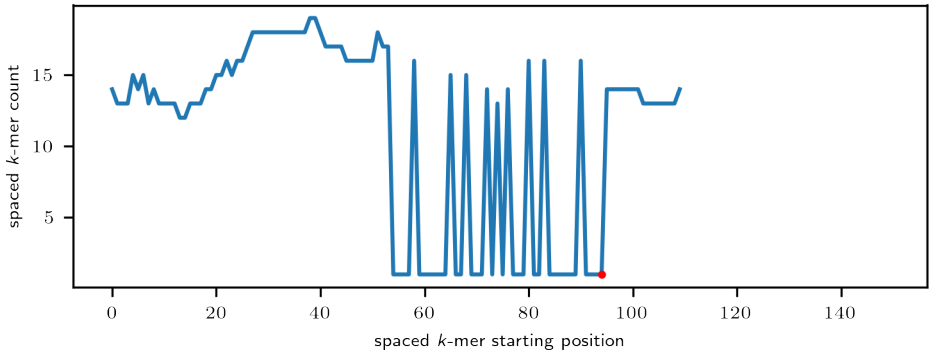


Figure 3.1.: Spaced k-mer count profile. Example profile for a sequence of length 150 and a spaced k-mer pattern of length 41 with weight 32, $P = 11110111111011011101010111011011111101111$. The error position located at position 94 (red circle) leads to a deviation in the count values of all spaced k-mers overlapping that position with an informative position, but not for the ones that overlap the error with a non-informative position.

Based on the counts of the spaced k-mers existing in a set of sequences, it is possible to determine for each position in a sequence how often the k-mer starting at that position occurs in the entire set of sequences. This can be visualized with a spaced k-mer count profile (see Fig. 3.1). Each data point in the count profile for sequence S corresponds to $count(K_i)$ with K_i being the spaced k-mer starting at position i in S. Obviously, the length of the spaced k-mer count profile is ℓ positions shorter than the sequence itself, as the last spaced k-mer of S is $K_{|S|-\ell}$.

If we consider a set of reads obtained from a sequencing experiment, we expect to have fully or partially overlapping sequences, correlating with the sequencing coverage. The count values of the spaced k-mers in a read should therefore reflect the local coverage of the genomic region this read originated from. Sequencing errors, however, do not follow the random sampling process but introduce new k-mers. Consequently, the frequency associated with these k-mers is lower compared to the true k-mers from the genome and sequencing errors can be identified as discrepancies in the count profiles.

Different error types would thereby lead to different deviations in the count profiles, see Fig. 3.2. A single substitution error would affect k spaced k-mer counts in the range of ℓ positions. If a spaced k-mer contains the sequencing error at an *informative* position, it is affected, whereas a k-mer containing the sequencing error at a *non-informative* position is not affected. Consequently, the drop in the spaced k-mer count profile follows the spaced pattern P. If the error is unique, it will drop to 1. However, if the sequencing error is an insertion, each spaced k-mer overlapping the respective error position is affected, independent of informative or non-informative positions. This is because each nucleotide after the error position in the respective ℓ-mer is offset by 1 compared to the true sequence. Consequently, ℓ count values are affected, and the count profile shows a continuous drop of length ℓ. The same applies to a single deletion error. Each spaced k-mer overlapping with the deleted position is partly offset by 1. The first spaced k-mer, which is not affected, is the one that starts directly after the deleted position. Therefore, a single deletion

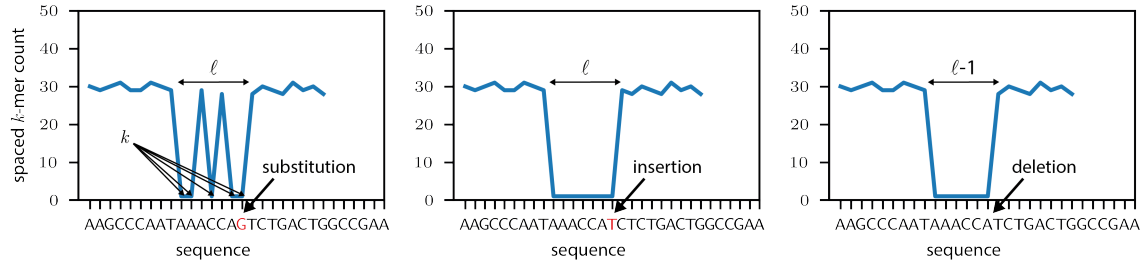


Figure 3.2.: Deviations in the count profile depending on the error type. Given the sequence S =AAGCCCAATAAACCACTCTGACTGGCCGAA of length 30 with a mean coverage of 30 and a spaced k-mer pattern $P = 1101011$ of length 7 and weight 5, different variants of the sequence at position 15 leads to different count profiles. From left to right: a single substitution error, leading to a drop following the spaced k-mer pattern; an insertion, leading to a continuous drop of length $\ell = 7$; and a deletion error, leading to a continuous drop of length $\ell - 1 = 6$.

error will affect $\ell - 1$ contiguous counts in the count profile. For multiple insertion errors in a row, $\ell + n$ counts are affected, where n is the length of the insertion. Further deviations from this pattern arise when an indel affects a repeated nucleotide.

Using k-mers to identify sequencing errors is quite common [Alic et al., 2016]. However, here we use the discrepancy along reads as a main source of information. This enables for a correction strategy based on the local k-mer frequency instead of a global threshold obtained from a k-mer spectrum analysis. A similar approach is also implemented in Pollux [Marinier et al., 2015] using continuous k-mers. However, the use of spaced k-mers allows for an error correction independent of errors in the immediate proximity (see section 3.1.2.2) and a direct distinction of the different types of errors (substitution, deletion, and insertion), in most cases.

3.1.2.1. Maximized spaced k-mer count profiles

As described above, a single substitution error leads to a drop in the count profile following the spaced k-mer pattern P. We expect each spaced k-mer that overlaps the error position with an *informative* position to have a low count. Using a maximization of all count values associated with spaced k-mers overlapping a certain position i in sequence S

$$maxcount(i) = \max_{i-\ell<j\leq i}(count(K_j)|K_j \text{ has an } \textit{informative} \text{ position at } i)$$

will therefore raise the count value of all affected positions to the neighboring level, except for the error position itself. This allows to identify the erroneous position directly from the maximized count profile. See Fig. 3.3.

3.1.2.2. Advantages of spaced k-mers

The main advantage of spaced k-mers over contiguous k-mers comes when correcting multiple sequencing errors in close proximity. The existence of more than one error within a range of k

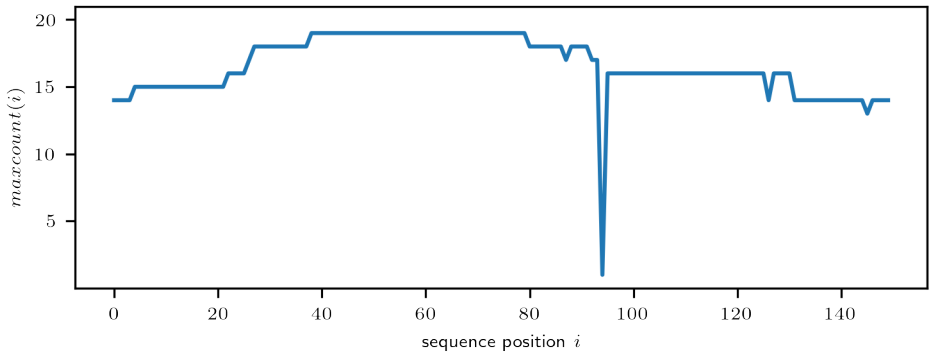


Figure 3.3.: Maximized spaced *k*-mer count profile corresponding to the spaced k-mer count profile shown in Fig. 3.1. The error at position 94 can be identified directly.

positions in the sequencing read would lead to a completely merged drop in the count profile when using contiguous k-mers, making it difficult to find the erroneous positions and to choose k-mers for the correction evaluation. Multiple consecutive k-mers would always overlap both sequencing errors, and a single substitution correction would never be able to correct them. Using spaced k-mers instead allows for a much higher resolution of the erroneous positions. Usually, spaced k-mers exist that have a non-informative position at both errors, as well as k-mers that only have an informative position at one of the error positions. This observation allows for (i) obtaining the true local coverage, (ii) finding the exact error positions with high accuracy and (iii) allowing for an independent error correction due to the selection of appropriate evaluation k-mers (see next section and Fig. 3.4).

3.1.3. Error correction strategy

As described above, potential erroneous positions can be identified as discontinuities in the spaced k-mer count profiles. In CoCo, I use the maximized spaced k-mer count profiles to identify potential substitution errors as drops and choose spaced k-mers around those positions for the error correction.

Thereby, I define a spaced k-mer K_i as potentially erroneous if the count is smaller than 1% of the maximal count in the immediate neighborhood (window of length ℓ centered around i) plus a small pseudocount (default: 1). Consequently, different counts are being seen as erroneous depending on the count values in the neighborhood. For example, for a neighborhood count of 5 a count of 1 would be seen as potentially erroneous, whereas for a neighborhood count of 100 all positions with counts ≤ 2 would be marked as potential errors, and for a neighborhood of 1000 counts ≤ 11 would be considered as potential errors. This leads to a threshold adapted to the local coverage within the read, instead of a globally set threshold. For reads originating from lower coverage genomes the threshold is dominated by the pseudo count, whereas in high coverage regions or in reads occurring from more abundant genomes the percent criterion is dominating. Thus, this strategy allows being aware of the wide variety of abundances occurring

in metagenomic samples and to adjust the error correction accordingly. As an exception, I never mark positions as errors within a neighborhood count < 5 (lowerbound criterion), as the absolute count difference is only very small and there is high probability that the count value is actually correct for this region.

For a certain read, I first pre-calculate the neighborhood counts (as defined above) and derive from these the neighborhood-specific threshold for the spaced k-mer count at each position in the read. The error correction procedure is then organized iteratively to process different kinds of errors, and an overview is given in Algorithm 3. First, I scan for substitution errors and try to correct them with high confidence using a very conservative two-side correction (lines 7-19). Afterwards, I scan for indel errors as well (lines 20-23). This order reflects the error type distribution in Illumina sequencing, where substitutions errors are the dominating type [Schirmer et al., 2016]. During the indel correction, I also consider substitution errors on the edges of a sequencing read (ℓ positions from the beginning/end) as they might have led to drops in the count profiles that differ from the expected pattern for a substitution error. For example, a single substitution error in the first few positions of a read might still show a continuous drop in the maximized count profile when no spaced k-mer exists that covers the previous positions, but not the error position itself. Thereby, both indel and "edge substitution" correction is performed using again a two-sided conservative correction.

For the first round of substitution error correction, I scan the whole maximized profile for positions with $maxcount \leq$ threshold. These positions are marked as potential errors and stored in an array. Additionally, all positions, which are affected by an erroneous position, i.e. the start positions of all k-mers that overlap the error position with an *informative* position, are marked to be affected by this error. This is represented by a bit vector for each position, where a set bit at a position j means that error j is overlapped by the k-mer starting at this position.

This allows for an efficient selection of evaluation k-mers for the subsequent correction step after all potential errors in the read have been marked. I iterate over the vector of potential erroneous positions: For each such position N, the two spaced k-mers K_l and K_r are identified that (i) are affected by the error to be corrected, (ii) are not affected by any other error and (iii) have a maximal distance of their starting positions in the read. These criteria were selected as evaluating all k-mers would be inefficient, whereas a single k-mer might not be informative enough to make an accurate correction decision. As a compromise, I choose the two k-mers for the correction evaluation that carry as much information from the remaining read as possible, while being independent of other errors. In the simplest case, K_l and K_r will be the k-mers that have the erroneous position as the first (leftmost), respectively the last (rightmost), nucleotide. Depending on the distance between errors and the spaced pattern, however, one or both of these k-mers might overlap another error and the next one has to be selected to ensure independence, see Fig. 3.4. A similar strategy to choose a leftmost and rightmost k-mer was also applied in Musket, but it does not check for independence of other errors [Liu et al., 2013].

```
1  lookup ← generate spaced k-mer lookup table
2  for r in reads do
3      p ← generate count profile
4      repeat
5          mp ← maximize count profile
6          np ← pre-calculate neighborhood thresholds
7          errors ← find drops in mp
                        /* potential substitution errors */
8          for N in errors do
9              affected ← mark all positions affected by N
10         end
11         for N in errors do
12             choose non affected evaluation k-mers K_l and K_r
13             explore possible substitution corrections at N
14             if valid(correction) and unique (correction) then
15                 apply(correction)
16             end
17         end
18         p ← update count profile
19     until corrections have been performed
20     errors2 ← find drops in p
        /* potential indel or edge substitution errors */
21     for N in errors2 do
22         do indel correction and substitution edge correction
            (details listed in the main text)
23     end
24     if number of correction > maxNumCorr then
25         revert corrections
26     end
27 end
```

Algorithm 3: Pseudocode of the iterative error correction strategy implemented in CoCo.

As the evaluation k-mers are independent of other sequencing errors, their counts should improve when the sequencing error is corrected successfully. I locate the nucleotide position associated with the potential error in K_l and K_r and replace the associated nucleotide with the three other possible ones. Afterwards, I evaluate if the count values associated with K_l' and K_r' improved, i.e. the new count value must no longer meet the criterion for an incorrect position. If both new count values of K_l' and K_r' rise over the error threshold, I consider the replacement as a possible correction. However, if multiple corrections for the same error position achieve this, no correction is performed. The decision to only make unique corrections avoids over-correction (false positives) and was made in the context of using CoCo as a preprocessing tool for PenguiN. It leads to a very conservative strategy that might miss some possible corrections (less sensitive), but should also minimize the risk to make spurious corrections. This is especially relevant if we encounter a position that could be changed to one strain or the other. In the context of PenguiN's assembly strategy, it is better to have a sequencing error than to accidentally create

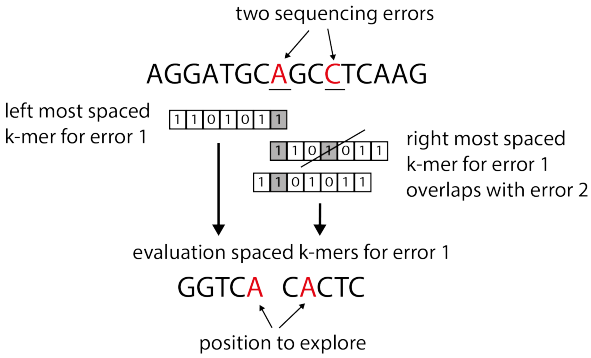


Figure 3.4.: Choosing evaluation k-mers in the case of errors in close proximity. In the simplest case, CoCo chooses the left- and rightmost k-mer that overlaps the error for the correction evaluation. However, if one of the spaced k-mer overlaps another error with an informative position, it is discarded and CoCo shifts its starting position to find the next k-mer that overlaps the error to correct with an informative position but has a non-informative position at all other error positions in close proximity.

a read that is a chimera between different strains and would then lead to misassembled contigs (see section 3.4). If a unique correction is found, the correction is applied by replacing the nucleotide in the sequence.

In principle, it is possible that for a certain error position no two distinct spaced k-mers can be found, which satisfy the conditions for the evaluation k-mers due to a very high number of errors in close proximity. Therefore, I perform the correction iteratively until the number of corrected errors matches the number of identified errors or no more confident corrections can be performed. However, in practice, I saw that usually one round is sufficient.

Whereas substitution errors are the dominating type of sequencing errors in Illumina sequencing, a small fraction of indel errors also need to be accounted for. However, due to the much smaller probability to have an indel error, the indel correction in CoCo was much more simplified. I just scan the "normal" count profile for drops and distinguish three cases: (1) a drop of length ℓ or $\ell - 1$ and boundaries on both sides, (2) a drop of length $\leq \ell$ without a left boundary, and (3) without a right boundary. Thereby, a drop starts when the count first drops below the local error threshold and ends when a count value is above it. I only consider single indels, as otherwise the number of possible corrections grows exponentially. So, if a drop is larger than ℓ, I do not perform any correction. In case (1), with i being the last positions with a low count, I consider an insertion error at position i and, if the drop is exactly of length $\ell - 1$, a possible deletion error between i and $i+1$. For the deletion correction, I explore corrections by inserting a nucleotide between i and $i + 1$ and choose the leftmost and rightmost spaced k-mer that contain the newly inserted nucleotide as evaluation k-mers. For an insertion error, I delete the nucleotide at position i and subsequently evaluate using three k-mers: The left- and rightmost k-mers that contain the position before and after the deleted nucleotide and a third one that is centered around the deleted position. I found choosing the third k-mer to be necessary in this case to consider the fact that the correction of an insertion error that has introduced the same base again (e.g. an 'A' after an 'A'), can only fully be evaluated with a k-mer that carries the information context from both sides.

In cases (2) and (3) the drop is only partly existent at the edge of the count profile, it could either be the result of an indel error or hint to a single substitution error close to the edges. Therefore, I first try a substitution correction on the last drop position before trying to apply indel corrections (as described above). For simplicity, I assume that no other errors are in close proximity any more (most substitution errors should already be fixed from the first step) and choose the evaluation spaced k-mers accordingly.

3.2. Implementation

I implemented the described algorithm in a new software tool, called CoCo. It consists of two parts: First, it generates a spaced k-mer count lookup, storing the frequencies over the whole set of sequences. Second, it iterates over each sequence of the set linearly, builds the count profile using the count lookup table and applies the error correction procedure. The whole workflow of CoCo is depicted in Fig. 3.5. A crucial step thereby is the efficient storage of the spaced k-mers and their counts.

The default pattern in CoCo is $P = 111101111110110111010101110110111111101111$. The length of P is 41 and the weight is 32. I choose the length that long to avoid common short repeats, which might otherwise confound the spaced k-mer count profiles and the weight as k=32 as it is the longest k that can be represented in a 64-bit word. A smaller number of informative positions would be less sensitive, and a higher number would not fit into a 64-bit word. I expect that the perfect choice depends on the genomic variation represented in the dataset as well as on the number and distribution of sequencing errors. The distribution of informative and non-informative positions in the pattern was selected for no specific reason as I do not expect the pattern itself to have much influence on the correction, as long as the non-informative positions are spread approximately equally (this was confirmed by the analyses done by Anton Farr during his internship project in our group). However, the inversion symmetry in the pattern was chosen on purpose, as it allows for a uniform handling of the reverse complements. For a symmetric pattern, the reverse complement of a sequence would simply lead to the reverse complement spaced k-mers, which is not true for a non-symmetric spaced pattern. So a k-mer and its reverse complement can be considered as the same object. Each time when a spaced k-mer is considered (when counting, accessing, ...) the canonical representation is used, i.e. of each k-mer the reverse complement is computed and between the two, the one with the lower index is used. For faster computation of the reverse complement, I use the Streaming SIMD Extensions (SSE) instruction set in CoCo, if available.

3.2.1. Efficient storage of spaced k-mer counts

The generation of spaced k-mer count profiles requires the storage of spaced k-mers and their counts over a whole set of sequencing reads. However, due to the large size and complexity of metagenomic samples, the number of k-mers and frequencies is usually too large to be hold in the

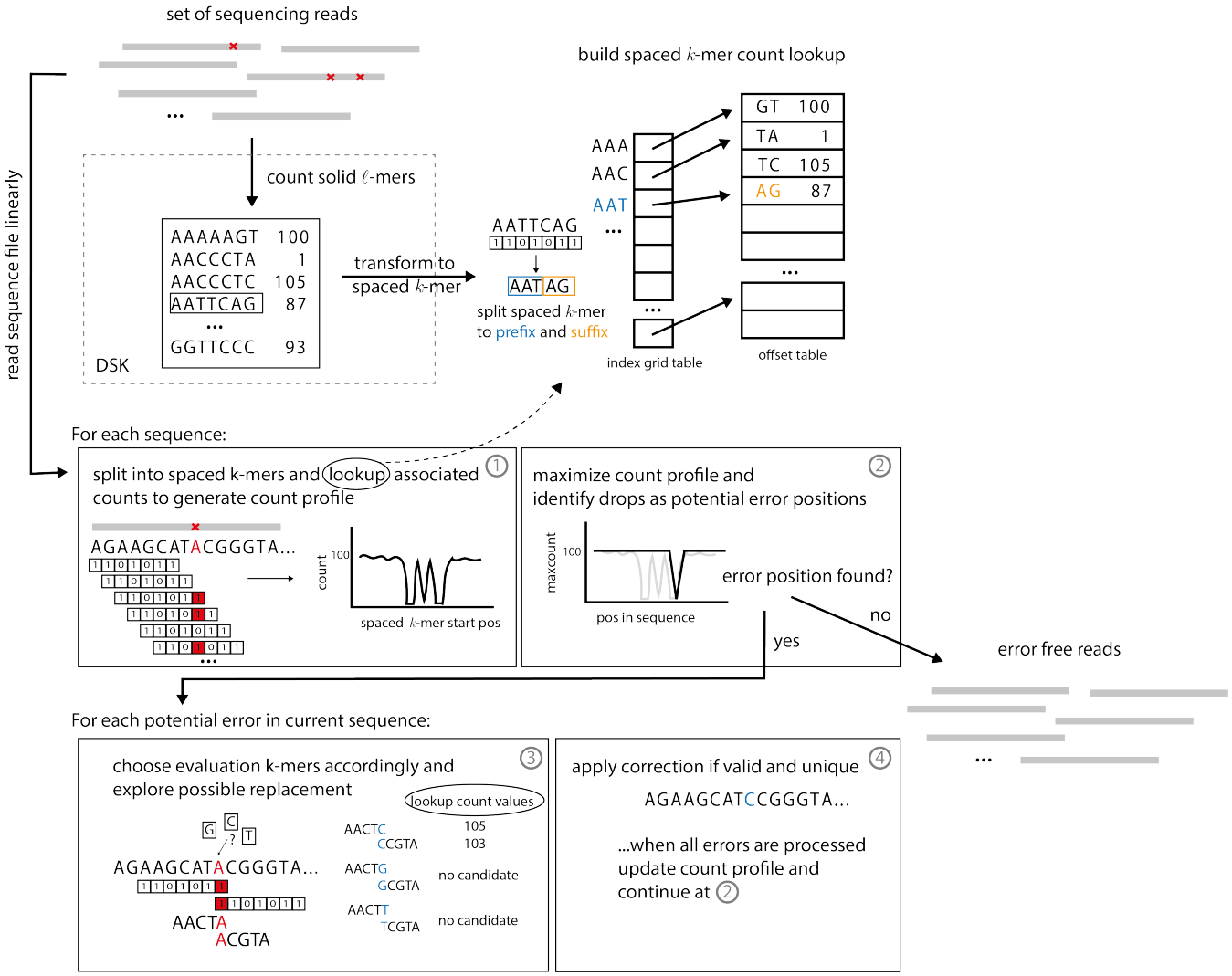


Figure 3.5.: CoCo workflow. As input, CoCo gets the sequencing reads (sequencing errors are marked with a red cross) and a pre-computed ℓ-mer count table generated by DSK. It then proceeds in multiple steps. First: transform contiguous ℓ-mers into spaced k-mers and store them together with their counts in an efficient lookup table (upper part). Second (lower part): parse sequencing file and generate for each sequence the spaced k-mer count profile using the count lookup table (1) and apply the error correction procedure (2-4). Here, only the substitution error correction is shown.

main memory of a modern computer, if a simple lookup of k-mer indices and their counts would be used. Further, the high number of lookups to build the spaced k-mer count profiles for each sequence (approximately one access per sequence position) requires the data structure (lookup table) that holds the counts of all spaced k-mers to be very space efficient while providing very fast access.

The data structure I use in CoCo consists of two tables, which can be held in memory: the index grid table and the offset table. For memory efficiency, I compress nucleotide information and represent them with a two-bit alphabet (A:00, C:01, T:10, G:11). Based on the two-bit alphabet, spaced k-mers can be represented with $2k$ bits. However, I only store the last I bits (suffices) explicitly. They are stored in the offset table – grouped by common prefixes - alongside

with the count associated with the spaced k-mer. The remaining p bits (prefix of the spaced k-mer) are only stored implicitly. In the index grid table, I store the starting positions for the common prefix blocks in the offset table, indexed by the first p bits. Thereby, no information is lost, as the full spaced k-mer can be reconstructed by $k\text{-mer} = (p << I)|(\text{last } I \text{ bits})$. See Fig. 3.5 for visual demonstration.

In the default configuration with $k = 32$, the spaced k-mer is divided into a 30 bit prefix and 34 bit suffix. This leads to a fixed memory requirement of 8 GB for the index table (2^{30} entries) plus 5 bytes per spaced k-mer and 2-4 bytes for its count. The complete memory requirement to store the count of M spaced k-mers would then be $8\,\text{GB} + M \cdot 9\,\text{B}$. This enables storing the complete spaced k-mer count information of samples with billions of reads in memory ($< 128\,\text{GB}$), while ensuring fast access, as only short blocks of suffixes have to be searched for the correct entry.

As (contiguous) k-mer counting is essential in many bioinformatics applications, a number of ready-to-use tools exist already, which are highly optimized, e.g. Jellyfish [Marçais and Kingsford, 2011], DSK [Rizk et al., 2013], KMC3 [Kokot et al., 2017], etc. Therefore, I decided to first count contiguous 41-mers with an external tool and use them as input for CoCo and internally translate them into the canonical spaced 32-mers. At the time of writing, CoCo supports the hdf5 file format as created by DSK [Rizk et al., 2013].

In the first step, CoCo reads the precomputed contiguous 41-mers and their counts from a hdf5 file using the gatb-core library [Drezen et al., 2014] and sets up the index grid table by translating the contiguous ℓ-mers to spaced k-mers on the fly. It determines for each (prefix) block the number of suffixes to store and saves each block's starting position in the offset table into the index grid table. Then, in a second iteration, it scans the hdf5 file again to fill the offset table. Based on the prefix, CoCo determines for each spaced k-mer the correct block in the offset table and stores the spaced k-mer suffix in the offset table. As different contiguous 41-mers can also lead to the same spaced k-mer (as shown in section 3.1.1), I check if the suffix of the spaced k-mer already exists in the selected block. If so, I do not add a new entry in the offset table, but only sum up the count values. If not, a new suffix is added in the offset table and the corresponding pointer in the index table is shifted by one. So the pointers in the index grid table always point to the next free space in the corresponding block in the offset table. After completely scanning the hdf5 file and filling the tables, I reset the pointers to the starts of the blocks.

3.2.2. Processing sequencing reads

In the second step, CoCo processes the sequencing reads linearly. For each sequencing read, it computes the corresponding spaced k-mer count profile and applies the error correction procedure individually (lower part of Fig. 3.5). As input, CoCo accepts sequencing reads in FASTA or FASTQ format (paired-end and single). The sequencing reads are parsed one by one using the kseq parser from Heng Li [Li, 2009] and the kseq wrapper functionality from MMseqs2 [Steinegger and Söding, 2017], which enables processing of compressed files. Then, CoCo iter-

ates linearly over the sequence. Thereby, it splits the sequence into 41-mers and obtains the corresponding spaced 32-mers using the specified spaced pattern. Then it stores the associated count from the lookup table in an array, representing the spaced k-mer count profile. If a sequence contains an 'N' or any other wildcard symbol, true spaced k-mers can still be obtained, whenever the 'N' affects a non-informative position. If the 'N' meets an informative position, there is no valid spaced k-mer representation according to the two-bit alphabet representation. However, an explicit representation is not necessary as there will also be no associated count value stored in the lookup table (DSK does not report 'N' containing 41-mers). In this case, the count values stored in the spaced k-mer count profile will simply be set to zero for these spaced k-mers. Consequently, an 'N' leads to the same drop pattern as a substitution error, but with counts of zero. CoCo will try to replace the 'N' with either A, C, G, or T in the subsequent error correction procedure.

Next, CoCo applies the error correction procedure itself, following the strategy described in section 3.1.3. It generates the maximized count profile and scans it for discrepancies to obtain potential substitution error positions. It selects the evaluation k-mers according to the strategy described on page 82 and tries to correct the substitution errors by improving the count above the neighborhood-specific error threshold. If a valid substitution is found and if it's the only one for that position, the correction is applied. If quality strings are given, additionally also the quality score is updated as the average of the quality scores adjacent to the corrected position. When the substitution error correction is completed, CoCo updates the count profile and applies the indel and edge substitution correction to it, as described above. Finally, one last attempt to correct remaining substitution errors is performed. The strategy thereby is the same as the previously described substitution error correction, but allows for the case that $K_l = K_r$, i.e. only using one evaluation k-mer when it is not possible to find two independent ones (one-sided correction). These corrections are much less confident, but correspond to edge cases. Note, independent on the stage, all corrections are only performed if there is one unique alternative nucleotide at a certain position and discarded otherwise.

In the end, the sequence is written to the output file using the same mode (FASTA or FASTQ) in that the input was provided. Then, the next sequencing read is processed. During the whole process, CoCo keeps track of the number of corrections performed to prevent over-correction. If a read underwent too many corrections (by default more than 10), the changes are reverted and the original sequence is output. Additionally, the statistics about corrections performed is also gathered globally and provided to the user in the end.

3.2.3. Further modules

CoCo is built very modular with a high level of abstraction. This allows for a simple replacement of individual steps, and therefore to build functionality other than the error correction procedure on top of the count profiles. During my doctoral research, I build further functionality for abundance estimation and to filter chimeric reads, as well as two further developer features to

output all spaced k-mer count profiles and to read out the spaced k-mer count table. However, these modules require further benchmarking yet.

3.2.4. Code availability

CoCo is implemented in C++ and the source code is freely available under the GPLv3-license. It can be downloaded from GitHub at https://github.com/soedinglab/CoCo. CoCo makes use of the gatb-core library [Drezen et al., 2014] and the kseq parser [Li, 2009]. The version used here for the benchmarks is GitHub commit 2737d13. To generate the pre-computed count table the counting software DSK [Rizk et al., 2013] has to be called beforehand. The DSK version used here for the benchmarks is GitHub commit 68b79e4 downloaded from https://github.com/GATB/dsk.

```
dsk -file raw1.fastq,raw2.fastq -kmer-size 41 -abundance-min 1 \
    -nb-cores 16 -out dsk.h5 -out-tmp dsk-tmp -max-memory 8000

coco correction -1 raw1.fastq -2 raw2.fastq --counts dsk.h5 \
     --outdir OUTDIR --outprefix OUTPREFIX
```

A documentation for CoCo can be found in section A.2 in the appendix.

3.3. Test Results

As the sequencing error correction approach in CoCo was originally designed and implemented in the context of the development of the de novo assembler PenguiN (see chapter 2), no comprehensive sequencing error correction benchmark was performed for CoCo as a standalone tool yet. However, in the following I present the results from the proof of concept analysis on reads simulated from a single viral genome and show some preliminary results for a mixture of two genomes, hinting at the potential of CoCo when compared to other tools.

For the data simulation I used WgSim [Li, 2011] with slight modifications compared to the customized version used in [Mitchell et al., 2020], see WgSim fork available at https://github.com/AnnSeidel/wgsim. In this version, WgSim also provides the error-free reads (true reads, a feature added by Mitchell et al.) and introduces random sequencing errors instead of recurrent ones. Having the true reads avoids the need for a mapping step for the evaluation of the corrected reads. Mapping the raw and the corrected reads each against the reference is common to evaluate error correction tools [Salmela and Schröder, 2011; Schröder et al., 2009; Yang et al., 2013, 2010], but can also be misleading. A read that originally stems from a genome A and is erroneously corrected in a way that it maps afterwards to a closely related genome B would be seen as correct and therefore lead to an incorrect assessment of accuracy. This becomes especially problematic in the presence of multiple strains. Having the true reads instead allows

for a direct comparison between raw reads (set of simulated reads with sequencing errors), true reads (set of reads with correct bases) and corrected reads (output of the correction tool to evaluate) and therefore for a more accurate and robust evaluation [Mitchell et al., 2020]. For the experiments presented here, I used the framework provided by Mitchell et al. that classifies each base into categories of true negative (TN), true positive (TP), false negative (FN), and false positive (FP) or trimmed. Thereby, it uses a multiple sequence alignment between the raw, true and corrected version of a read. TP describes correctly corrected error positions, FP stands for newly introduced errors, FN for non-fixed or incorrectly fixed errors and TN for an unaffected error-free base. The extra category trimmed contains the bases that were removed from the ends of sequencing reads by the error correction tool. All experiments were performed on the GWDG HPC cluster (www.gwdg.de) using individual nodes with two Intel Xeon E5-2640v3 processors at 2.6 GHz totaling 16 cores and 128 GB RAM.

3.3.1. Validation on simulated data

To provide a simple proof of concept analysis for the error correction strategy implemented in CoCo, I used data from a single *rhinovirus* genome (MN749156.1) and analyzed the spaced k-mer frequencies before and after the correction with CoCo (see Fig. 3.6). Thereby, I applied CoCo with default settings, meaning a spaced k-mer pattern with length 41 and weight 32 with the pattern as shown above. I simulated one dataset with a mean coverage of 30 and another one with a mean coverage of 1000, each with a sequencing error rate of 0.1% (only substitution errors).

Despite the small error rate, I obtained a large number of unique or near unique spaced k-mers before the correction, as can be seen in Fig. 3.6. This can be explained by the fact that each error introduces (approximately) k artificial k-mers. For the 30x coverage dataset, I obtained 4633 spaced k-mers that have a frequency of 1-4, for the 1000x coverage I obtained 145 620 (the extreme left side of the graph). After correction, for both datasets, the number of unique and near-unique spaced k-mers massively decreased (first peak basically disappeared) and the average spaced k-mer count increased (slight shift to the right for the true spaced k-mers). This illustrates that the process by which an error had turned frequent k-mers into infrequent ones is successfully reversed by CoCo's correction.

For the high coverage dataset only two spaced k-mers with a frequency lower or equal to 4 remain, for the low coverage dataset 88 remain. However, evaluating the set of CoCo corrected reads with the evaluation pipeline from Mitchell et al. against the true reads obtained from the customized WgSim simulation, showed that none of them was an overlooked sequencing error. Instead, CoCo correctly distinguished between artificial k-mers with low counts resulting from sequencing errors and true k-mers with low counts, resulting from the random sampling process of read origins.

Next, I also analyzed CoCo's strategy in more detail, inspecting the spaced k-mer count profiles before, during and after the correction runs. Fig. 3.7 visualizes the spaced k-mer count profiles

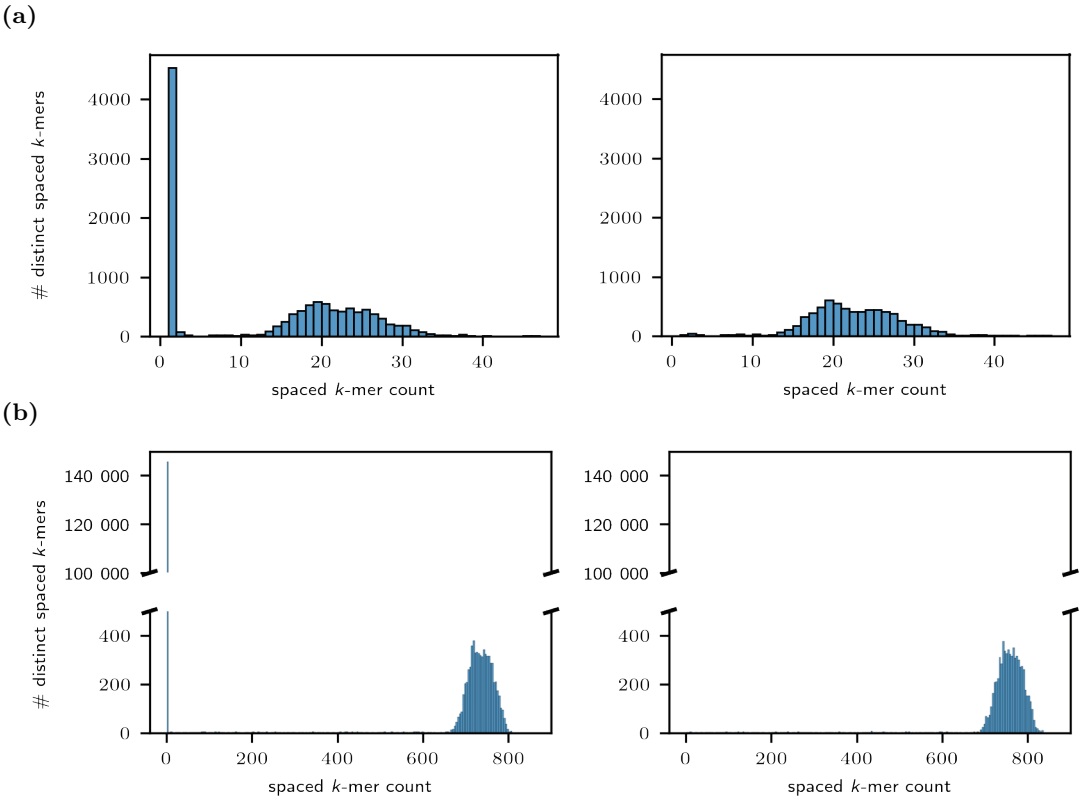


Figure 3.6.: Spaced *k*-mer count histograms for the single rhinovirus dataset simulated at different coverage values. The x-axis refers to the spaced k-mer count, i.e. it shows the number of times a spaced k-mer occurs in the data, the y-axis refers to the frequency this count is observed. **(a)** shows the distribution for the 30x coverage dataset for the raw reads (left) and for the CoCo corrected reads (right). The first peak results from unique or near-unique spaced k-mers, which result from sequencing errors. The main peak of the bell shape occurs around x=21. After CoCo's correction, the first extreme peak disappeared as the number of unique and near-unique spaced k-mers massively decreased. The main peak shifts slightly to the right (x=22), as each now corrected spaced k-mer increases the count of the true ones by one. **(b)** shows the distribution for the 1000x coverage dataset for the raw reads (left) and for the CoCo corrected reads (right). Also here the first extreme peak on the raw reads disappeared after CoCo's correction and the mean of the bell-shape shifts from x=715 to x=738.

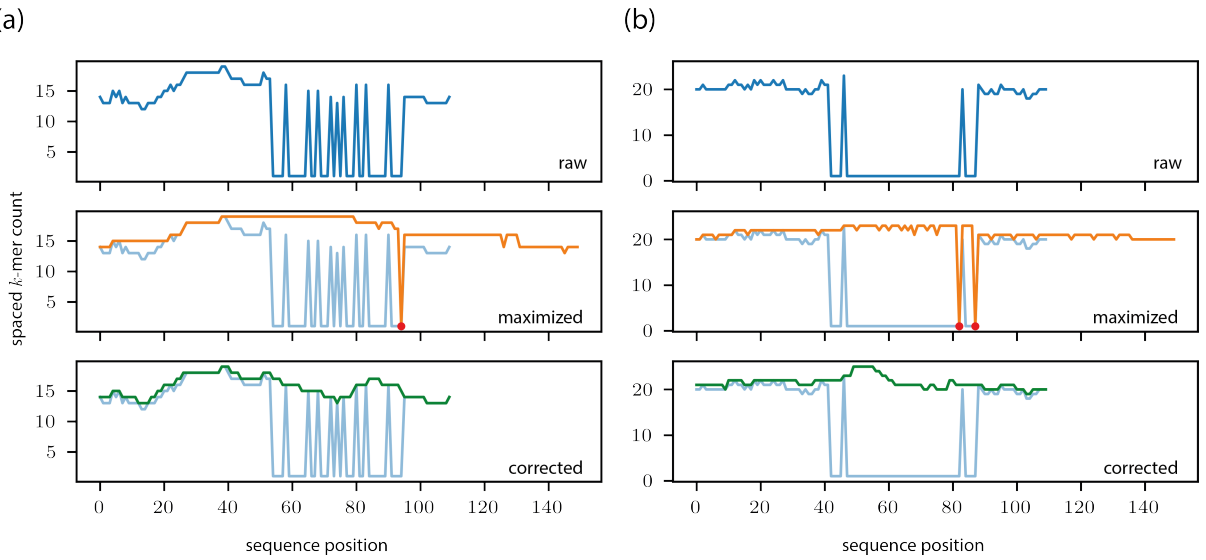


Figure 3.7.: Spaced *k*-mer count profiles from the 30x coverage dataset of a single rhinovirus. Two count profiles from **(a)** a read with a single substitution error and **(b)** a read with two errors in close proximity. The top row shows the "raw" count profile, as calculated using CoCo's lookup table before the correction. Significant deviations show that errors are present in the read. The orange line in the middle row shows the maximized count profile, which CoCo calculates during its process, directly revealing the positions to correct (red dots). The green line in the bottom row shows the count profile of the read after CoCo's correction. The previously observed significant deviations are now gone, hinting at a successful correction.

for two examples from the 30x coverage dataset, a sequencing read containing a single sequencing error (same profile as used above for the algorithm explanation) and a sequencing read containing two nearby errors. This illustrates that the theoretical assumptions that have been made for the algorithm are also visible in practice. The error positions can be identified directly from the maximized count profile, allowing for an accurate correction which results in a count profile without significant deviations after CoCo has processed the read.

Next, I increased the error rate during the simulation to 0.3%, 0.5% and 1%, respectively. This obviously increased the total number of sequencing errors in the raw reads, as well as the probability for multiple sequencing errors within one read. Whereas the 0.1% error rate, used before, had led to reads with 0-2 errors, I observed reads with 0-4 errors for the error rate of 0.3%, 0-5 errors for the error rate of 0.5% and reads with up to 7 errors for the 1% error rate. The total number of reads with any error before the correction was approximately 13%, 36%, 53% and 76%, respectively. Applying CoCo on these data massively decreased the overall number of sequencing errors and significantly increased the fraction of error-free reads across all datasets. The results are presented in Fig. 3.8.

For the four datasets, the fraction of error-free reads increased from 87% to 100%, from 64% to 99%, from 47% to 97% and from 24% to 86%. Thereby, also the majority of reads with multiple sequencing errors were successfully corrected.

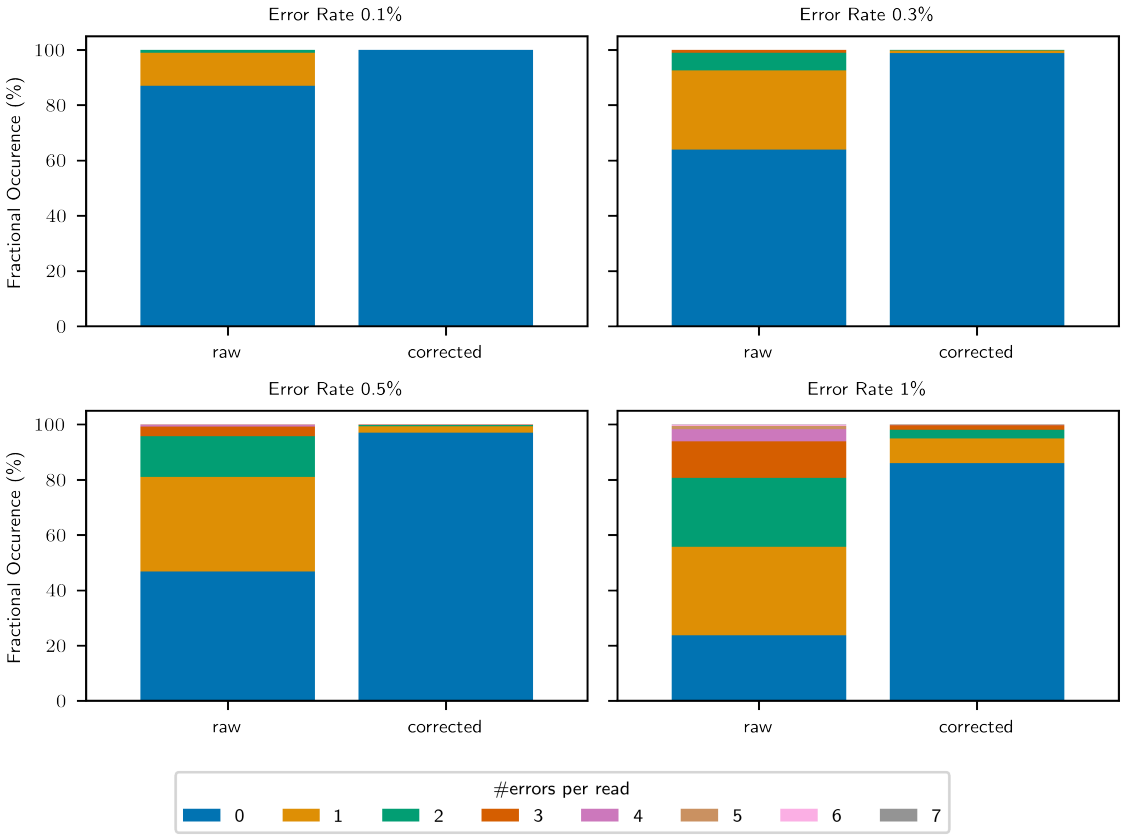


Figure 3.8.: Fractional occurrence of erroneous reads before and after the correction for different simulated error rates. For all four error rates tested (0.1%, 0.3%, 0.5% and 1%) CoCo massively increased the fraction of error-free reads (blue). Thereby, CoCo was also able to massively reduce the fraction of reads with multiple errors. The slightly worse performance for the 1% error rate dataset could probably be improved by choosing a spaced k-mer pattern with less densely packed 1s, i.e. a decreased weight.

3.3.2. Comparison with other methods

Next, I evaluated how CoCo performs on a mixture of genomes. I used an Escherichia phage T4 genome (MT984581.1) and simulated reads from the original reference as well as from an alternative reference, using WgSim's internal polymorphism feature with -r 0.05 -R 0.15, i.e. a divergence of 5% with a 15% fraction of indels.

To investigate thereby also the influence of the ratio between low and high abundant genome, I simulated read sets with different coverage values and mixed them in four different ratios: 10:10, 10:50, 10:100, 10:500. Thereby, the first value always indicates the coverage of the mutated reference genome and the second value the coverage simulated for the original reference. I also tried the same ratios with higher absolute coverage values. However, higher values for the coverage turned out to be very costly for the evaluation framework and were therefore finally left out. All simulations were run with an error rate of 0.1% and created paired-end reads with 2x150 bp and an outer distance of 260 with a standard deviation of 20.

I compared the results of CoCo with those obtained from several other error correction tools including Bcool (v1.0.0) [Limasset et al., 2020], BayesHammer (v3.15.2) [Nikolenko et al., 2013], Fiona (v0.2.10) [Schulz et al., 2014], Musket (v1.1) [Liu et al., 2013] and Pollux (v1.0.2) [Marinier et al., 2015].

As all of these programs use k-mers in some way, but have different restrictions to set the k-mer size e.g. only odd or up to a specific k value, I tried different settings for a fair comparison, except for Fiona, which adapts parameters depending on the data automatically [Schulz et al., 2014]. However, the results for the different settings tried did not differ much. Therefore, I only report the results for the default settings in the following. All tools were set to utilize all 16 cores, if they had a multi-threading option. On the 10:50 coverage dataset, BayesHammer did not run successfully, instead ending with a segmentation fault. The exact cause could not be determined, and therefore results for BayesHammer on this dataset are missing in the following.

A summary of the results can be found in Table 3.1. Thereby, sensitivity and precision were calculated using the evaluation pipeline from Mitchell et al. [Mitchell et al., 2020]. They are defined as followed:

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

Further, I calculated the fraction of reads with any sequencing error as the number of reads with at least one base that was classified as FN or FP, divided by the total number of reads.

All tools show high sensitivity values $> 90\%$ across all datasets. Fiona reaches the highest values for the per-base sensitivity on all four datasets. However, the corrected read set obtained from

Table 3.1.: Error correction results on the T4 phage mixture datasets.

(a) genome coverage 10:10

Tool	Reads with Error (%)	Base Sensitivity (%)	Base Precision (%)	Time (s)	Memory (MB)
Original	13.86	-	-	-	-
BayesHammer	3.61	93.86	77.45	4	796
Bcool	1.20	98.33	87.00	9	587
CoCo	**0.69**	96.15	**97.61**	21	8418
Fiona	27.68	**99.00**	14.13	109	794
Musket	1.31	98.92	84.75	**1**	85
Pollux	6.61	91.09	49.11	18	**47**

(b) genome coverage 10:50

Tool	Reads with Error (%)	Base Sensitivity (%)	Base Precision (%)	Time (s)	Memory (MB)
Original	13.95	-	-	-	-
BayesHammer	-	-	-	-	-
Bcool	0.94	98.98	87.92	15	607
CoCo	0.51	96.89	**98.96**	39	8423
Fiona	13.83	**99.43**	15.93	140	842
Musket	**0.49**	98.86	95.44	**3**	92
Pollux	10.00	95.14	23.57	40	**80**

(c) genome coverage 10:100

Tool	Reads with Error (%)	Base Sensitivity (%)	Base Precision (%)	Time (s)	Memory (MB)
Original	13.87	-	-	-	-
BayesHammer	2.61	95.20	82.11	13	867
Bcool	0.64	98.83	91.96	19	528
CoCo	**0.43**	97.86	**98.27**	62	8426
Fiona	9.05	**99.74**	17.66	114	896
Musket	0.73	98.72	92.02	**5**	**122**
Pollux	6.98	96.03	27.60	47	116

(d) genome coverage 10:500

Tool	Reads with Error (%)	Base Sensitivity (%)	Base Precision (%)	Time (s)	Memory (MB)
Original	13.9	-	-	-	-
BayesHammer	1.40	95.50	91.79	46	1857
Bcool	0.40	99.04	94.94	21	694
CoCo	**0.39**	98.44	**97.03**	252	8439
Fiona	1.96	**99.94**	48.97	209	1301
Musket	0.41	98.86	95.88	**13**	**231**
Pollux	1.67	97.99	64.20	100	264

Fiona still showed a high proportion of erroneous reads. On the 10:10 dataset, the fraction of reads with errors after the correction was even higher than in the raw sequencing reads. This is due to a high number of newly introduced errors (as is also reflected in the precision value). Considering the sequencing reads in more detail showed that Fiona actually reverts the differences between the two genomes and not only the sequencing errors ("strain flipping") and that these bases constitute a large fraction of its false positives. Similar, but to a less extent, seems to apply for Pollux, the only other tool besides CoCo, which uses k-mer count profiles along a read.

Notable is that in each of the experiments, CoCo had a significantly higher precision than all the other tools tested. This is especially relevant, as a corrector should not introduce new errors. CoCo constantly showed a precision that is 1-11 percent points higher than the next best tool. Furthermore, CoCo was the best tool to generate completely error-free reads. It reaches the lowest fraction on the 10:10, 10:100 and 10:500 mixtures and was close to the best on the 10:50 dataset.

Besides the correction evaluation, I also compared the performance of the tools in terms of runtime and memory footprint. The results from Table 3.1 show that CoCo has a significantly higher memory peak than the other tools tested. This is due to the index table, which always occupies 8 GB of memory (see section 3.2.1), which clearly dominates CoCo's memory consumption here. As the dataset presented here is rather small, the theoretical advantage of this data structure cannot be seen. However, it can be seen that CoCo scales well with the coverage value, as its memory footprint does not increase across the datasets, while e.g. BayesHammer needs ~ 1 GB more memory between the 10:10 and the 10:500 dataset. Therefore, it is likely that with more complex data sets, the advantage of CoCo's lookup table will become apparent. Moreover, for small datasets, the use of the hash table instead of the lookup table could be considered (discussed in section 3.4). CoCo (including the DSK counting step) was also slower than the other tools tested, except Fiona. Thereby, the obvious bottleneck is the linear processing of reads, as CoCo is not yet parallelized. However, the total run time was still small due to the small size of the datasets and CoCo is not yet parallelized.

To quantify the overall performance of the error correction tools, I used the gain metric [Yang et al., 2013, 2010]. It is a summary of sensitivity and precision, calculated by

$$\text{Gain} = \frac{\text{TP} - \text{FP}}{\text{TP} + \text{FN}} \tag{3.3}$$

For a method that removes all errors without introducing new ones, the gain reaches one. When more errors are introduced than corrected, the value becomes negative.

Fig. 3.9 shows the gain metric for both phage genomes individually. As expected, each error correction tool reaches a higher base gain for the high abundant genome than for the low abundant genome. The differences in the base gain for the two genomes from the even mixture (10:10), could not be clarified. However, for all mixtures, CoCo is in line with the other tools

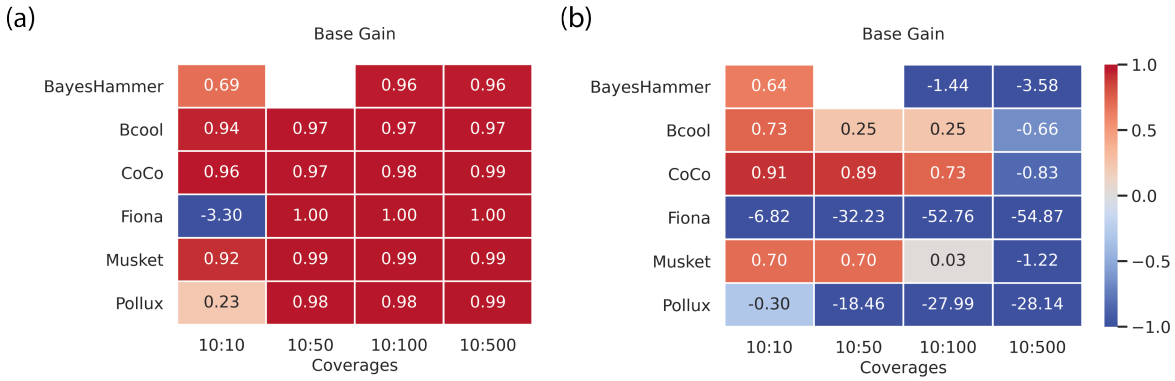


Figure 3.9.: Heatmap depicting the base gain across the different coverage ratios of the T4 phage mixtures. Each row corresponds to an error correction tool, and each column corresponds to a dataset with the given coverage combination of the two T4 phage genomes. **(a)** shows the base gain for the original genome, which is always the higher covered one in the uneven mixtures. It is represented by the second value of the coverage combination. **(b)** shows the base gain for the 5% diverged genome, which is always the lower covered one in the uneven mixtures. It is represented by the first value of the coverage combination.

for the higher abundant genome (see Fig. 3.9a), whereas it reaches significantly higher gain values for the low abundant one than the other tools (see Fig. 3.9b). The majority of tools reach either very small values or even a negative gain for the lower abundant one, indicating a high number of false positives, possibly resulting from strain flip corrections. The only other exception besides CoCo was Bcool, which shows the second-best overall performance on the lower abundant genome.

CoCo was the only tool that consistently showed a gain ≥ 0.7 on the 10:10, 10:100 and 10:100 coverage datasets. Only for the 10:500 mixture, it reached a negative gain for the lower abundant genome, as did all other tools. I argue, that at this coverage ratio, CoCo also starts to flip some SNPs between the different genomes due to the 1% drop criterion. This could probably be avoided by choosing a lower percent drop criterion or using another error detection criterion (discussed in section 3.4).

3.3.3. Impact on de novo assembly

To evaluate the effect of CoCo's error correction on assembly, I ran the PenguiN assembler on the set of uncorrected reads and on the corrected reads of the T4 phage mixtures. Thereby, I used PenguiN with 5 and 10 iterations on the amino acid level and nucleotide level respectively and default parameters otherwise. The results are shown in Table 3.2. We see that on all four mixtures, the assembly significantly improves when providing the CoCo corrected reads to PenguiN instead of the uncorrected ones. The assembly becomes more contiguous in view of number of contigs, length of the largest contig, average contig length and N50 value.

Besides this, the results shown during PenguiN's benchmark in the previous chapter, had demonstrated that CoCo can improve the accuracy of PenguiN's assembly also on real sequencing data.

Table 3.2.: Assembly results on the T4 phage mixture datasets using PenguiN with and without read correction.

	Number of contigs	Largest contig (bp)	Average Length (bp)	N50
10:10				
Uncorrected	127	12 621	2754.5	3456
CoCo corrected	30	99 454	28 359	48 682
10:50				
Uncorrected	118	173 996	19 590	49 600
CoCo corrected	30	168 900	41 151	95 144
10:100				
Uncorrected	135	113 262	19 838.3	49 087
CoCo corrected	39	168 569	31 144.5	56 687
10:500				
Uncorrected	176	127 798	17 946.3	47 923
CoCo corrected	62	168 914	25 408	95 157

On a mock dataset comprised of six *Caudovirales* genomes using CoCo pre-assembly yields final contigs with 8.75 mismatches per 100 kbp, and 3.98 indels per 100 kbp when compared to the reference genomes (0.013%), instead of 72.31 mismatches and 10.85 indels per 100 kbp (0.18%) when using PenguiN without error correction (see section 2.4.3).

Further, the results obtained on the real metatranscriptomic samples from activated sludge and aquatic environments used in the PenguiN benchmark suggest that CoCo's error correction also works on very complex metagenomic samples (see section 2.4.4). Applying CoCo pre-assembly results in even more complete phage genomes in PenguiN's final assemblies (increasing from 793 to 833). This once again demonstrates that CoCo's correction leads to less fragmentation in the assembly due to cleaner overlaps.

3.4. Discussion and Outlook

In this chapter, I introduced the new sequencing error correction tool CoCo. It detects sequencing errors as sudden drops in the sample-wide frequency of spaced k-mers along a read. It utilizes a new data structure for the efficient storage of millions to billions of spaced k-mers, allowing for the processing of large and complex datasets. For the correction itself, I decided for a very conservative strategy to minimize the risk of introducing new errors or loosing true diversity (false positives) due to over-corrections, which would in turn hamper subsequent de novo assembly instead of supporting it.

The notorious problem of error correction involves identifying and correcting errors that were introduced during the sequencing process. Other biases can be introduced, for example during sample processing and storage, amplification, or library preparation stages [Ma et al., 2019], but

go beyond the scope of this thesis and are therefore not discussed here further. Even though today's Illumina short read sequencing produces highly accurate sequences with typical error rates of 0.1-0.25% (sometimes also reported as 0.1%-1% depending on the sequencing platform) [Fox et al., 2014; Pfeiffer et al., 2018; Stoler and Nekrutenko, 2021], sequencing errors can significantly influence the assembly [Heydari et al., 2017; Kelley et al., 2010; Liao et al., 2019; Salzberg et al., 2012].

Most of the de novo assemblers used today employ de Bruijn graphs. Such graphs represent k-mers occurring in the set of input reads as nodes and the overlaps between them as edges, contigs can then be obtained as paths through the graph [Compeau et al., 2011a; Pevzner et al., 2001]. As each single sequencing error leads to k erroneous k-mers (i.e. k-mers containing at least one sequencing error), they complicate this task, leading to spurious dead ends, bulges and chimeric connections [Liao et al., 2019; Miller et al., 2010; Zerbino and Birney, 2008]. Furthermore, erroneous k-mers can easily outnumber the true k-mers, leading to growing de Bruijn graphs and therefore to substantially higher memory usage [Li et al., 2015, 2012]. To reduce these effects of sequencing errors, many assemblers use some kind of explicit or implicit correction procedure, removing low-frequent k-mers under a certain frequency threshold or employing the topology of de Bruijn graphs [Bankevich et al., 2012; Li et al., 2015; Liao et al., 2019; Pevzner et al., 2001; Zerbino and Birney, 2008]. Additionally, some de Bruijn graph assemblers even have their own error correction tool integrated, e.g. the SPAdes assemblers have the error correction tool BayesHammer [Nikolenko et al., 2013] built in. For overlap-based assemblers, sequencing errors are usually less problematic, as the pairwise overlap computation usually tolerates a minor fraction of mismatches, and sequencing errors can be eliminated during the consensus step [Baaijens et al., 2017; Li et al., 2012]. However, these assemblers might also benefit from cleaner overlaps, allowing for more stringent alignment parameters.

The main motivation for the project outlined in this chapter was to develop a sequencing error correction tool that can be applied pre-assembly to PenguiN (chapter 2), to support it in assembling more contiguous due to cleaner read overlaps and allowing for higher contig quality in the end. As we aim to deal with high (viral) population diversity in metagenomic samples with PenguiN, a sequencing error correction tool applied pre-assembly to PenguiN (1) has to be aware of uneven coverage due to the mixture of genomes with widely different abundances in a metagenomic sample and (2) must not cancel out true diversity resulting from different strain variants.

Similar to other sequencing correction tools, CoCo makes use of k-mers to identify and correct sequencing errors. However, it explores k-mer frequencies along the reads and identifies potential erroneous positions as discrepancies in those frequencies. In contrast to k-mer spectrum correctors, which distinguish between "solid" and "weak" k-mers based on a certain threshold, this allows for a local instead of a global distinction of potentially erroneous k-mers. I argue that this is much more suitable for a mixture of genomes / metagenomic samples, as it considers the context of a read. In contrast, a global threshold would cause a corrector to assume that the low coverage k-mers derived from low covered genomes are erroneous and causes it to change

them to k-mers from higher covered genomes. This seems to be the main drawback for many existing error correction tools, originally developed for genomic data.

The idea to maintain the k-mer frequencies along the reads as main source of information has already been previously implemented in a tool called Pollux [Marinier et al., 2015]. However, the implementation in CoCo differs substantially due to the use of spaced k-mers. As I have pointed out in section 3.1.2.2, using spaced k-mers together with the maximized count profile allows revealing the erroneous positions (resulting from substitution errors) directly and therefore also to identify errors in close proximity. This in turn allows for a better selection of evaluation k-mers, as different sequencing errors can be considered independently. Furthermore, spaced k-mers yield a better local estimate of the true read coverage value, as there are usually at least a few spaced k-mers which overlap the sequencing errors with non-informative positions. However, it remains to be noted that spaced k-mers do not give an advantage for correcting indels. Because of the shift within a k-mer due to a deleted or inserted position, the erroneous position cannot be masked out and spaced k-mers just behave as "normal" contiguous k-mers, leading to a continuous drop in the k-mer frequency profiles. However, as substitution errors are the dominating type in Illumina sequencing errors [Schirmer et al., 2016], I think CoCo's error correction strategy, which is focused on substitution errors and corrects insertions and deletions in a simple manner, should suffice.

In order to prove the general working scheme of CoCo, I have done several benchmarks on a single genome and have proven its applicability even in cases with multiple sequencing errors within one read. Further experiments have shown that CoCo reaches very high precision in the presence of closely related genomes, with a slight loss in sensitivity compared to state-of-the-art correction tools (see section 3.3.2). This together results in CoCo outperforming the other tools in terms of error-free reads after the correction. This is especially due to CoCo's performance for the lower abundant genome, as most of the other error correction tools show a high number of false positives due to "strain flipping". In contrast, CoCo can accurately distinguish sequencing errors from SNPs due to its local coverage estimation and error detection.

I think there are two main sources for the slight loss of sensitivity. The first source is obviously the lower bound criterion for the neighborhood (default 5), the threshold under which CoCo does not try to correct anything but just skips the region. With a read coverage of 10, we would expect a mean k-mer coverage of ~ 7 and expect to have at least some regions with a count < 5. Sequencing errors in these regions would then be systematically overlooked by CoCo. The second source is the very conservative choice to apply a correction. CoCo just discards a correction and leaves the potential sequencing error position unchanged if multiple corrections are possible. In principle, it is possible to improve up on that to obtain higher sensitivity values. For example, one could decrease the lowerbound criterion, use more k-mers to be evaluated, try to formulate a criterion which correction best matches the neighborhood count or just choose one out of the possible corrections in a consistent manner. All these would then lead to a more aggressive flipping of nucleotides and should therefore also increase sensitivity. However, to the best of my understanding a more conservative strategy with a slight loss in sensitivity is more

suitable in the context of de novo assembly, the main motivation for the conception of CoCo, especially if the true diversity should be maintained. An overlooked sequencing error should be much less problematic than a systematically and erroneously corrected nucleotide. In addition, it was previously shown, that more aggressive error correction tools with high sensitivity do not necessarily lead to the better assemblies [Heydari et al., 2017]. Instead, the loss of true k-mers seems to be much more problematic as they lead to breakpoints in the assembly or to chimeric connections, if the k-mer is wrongly replaced by a k-mer that actually occurs elsewhere in the genome or in another genome [Heydari et al., 2017].

At the current state, it remains to be seen how CoCo improves the quality of different de novo genome assemblers, also in comparison to other error correction tools. However, in association with the PenguiN assembler, my experiments already show substantial improvements. Applying CoCo prior to the assembly with PenguiN leads to a more contiguous assembly and higher accuracy in the final contigs. This could be shown on simulated and real datasets.

3.4.1. Shortcomings and further benchmarks

As mentioned above, the error correction in CoCo was designed and implemented in the context of the PenguiN assembler. Thereby, the general working scheme could already be proven and the beneficial impact to the PenguiN assembler was shown. Nevertheless, a comprehensive benchmark for CoCo as a standalone tool in comparison to other correctors is lacking at the time of writing. So far, CoCo was only compared to other correction tools on one small dataset (T4 phage mixtures). It remains to compare CoCo with existing tools on larger and more complex datasets, which would actually also represent the use case CoCo was designed for, in a better way. I could already show that CoCo was able to process a large and complex metatranscriptomic dataset. Besides the correction performance itself, it would be especially interesting to see thereby how the resource demands behave for such datasets. I think CoCo substantially benefits from the lookup table, whose advantages could not be shown on the small dataset. Additionally, CoCo was not compared yet to other tools regarding its improvement for de novo assembly. Whereas the interplay of CoCo and PenguiN was shown, a comparison to another correction tool before PenguiN's assembly is missing. Moreover, I would also be interested to see how CoCo would influence the performance of other de novo assemblers.

3.4.2. Algorithm related-extension

CoCo already shows promising performance. However, one could think of further improvements to CoCo. Foremost, I would like to increase the user-friendliness of CoCo by integrating the k-mer counting step itself directly into the software tool, instead of requiring the use of an external tool. Currently, I utilize DSK [Rizk et al., 2013] as an external tool for the counting step and provide its output as input for CoCo. However, there are newer and probably more efficient tools available, e.g. KMC3 [Kokot et al., 2017], Gerbil [Erbert et al., 2017] and KCMBT

[Mamun et al., 2016]. Comparing them regarding CoCo's use-case would allow finding a tool which is highly efficient and can be integrated into CoCo's software. This would then streamline the error correction process. From a developer's point of view, it might also be interesting in this context to get rid of the gatb-core library [Drezen et al., 2014] dependency, which is currently only necessary to parse the hdf5 file format DSK generates. Furthermore, there is potential to improve up on speed and memory. Currently, CoCo processes all sequencing reads linearly. However, this could be efficiently parallelized, as CoCo's processing of one read is completely independent of all other reads. This would then greatly benefit CoCo's speed and improve scalability further. Regarding the memory, there is already work in progress. Whereas, I expect the lookup table to scale nicely for large and complex datasets, it generates a huge overhead for small datasets. Therefore, I already considered another storage opportunity for the spaced k-mer counts based on a hash table instead of the lookup table for smaller datasets, where the 8 GB of the lookup table is an overhead. However, an analysis between the two data structures is still remaining. Adding a logic to estimate the number of different spaced k-mers upfront and deciding then automatically for one of the two storage opportunities could improve CoCo's capability to adjust time and memory requirements to the data. Additionally, the exact choice of the weight of the pattern can have a huge influence on the memory requirements. Currently, CoCo's default with 32 informative position leads to the need to represent 64 bits, of which 34 bits are saved as the suffix. Due to the default padding to full bytes by the compiler, this is inflated to 5 bytes, i.e. 40 bits. Using a pattern with 31 informative positions, however, would reduce the suffix to 32 bits, which would need only 4 bytes. At the same time, it would most likely not influence the sensitivity greatly. Therefore, this is something to consider when benchmarking CoCo on larger data sets in the future. From a more theoretical viewpoint, it would also be interesting to consider the use of multiple spaced k-mer patterns. However, this would come with the price of runtime and memory. Also, the handling of paired-end reads could be improved. Currently, CoCo can handle single-end and paired-end input files, but always considers each read individually. In the overlap regions, however, CoCo could potentially make use of the information from the mate pair when making a correction decision. This could improve on the sensitivity and precision of these regions and prevent the pairs from drifting apart due to the correction. Thereby, and also in general, it could be worthwhile to take also the quality scores available in FASTQ files into account. Finally, the identification and correction step itself could also be improved further when better statistical considerations are included. I already discussed that using a criterion for the detection that evaluates the local coverage along a read has several advantages over a static cutoff. A certain count can be considered as sequencing error for one read originating from a highly abundant genome, whereas the same count can be considered as true in low covered regions or rarely abundant genomes to prevent false positives. However, the choice of a percent criterion is still naive and might not be ideal, as certain coverage ratios would always be considered as errors. As an alternative, one could calculate a probability distribution to estimate the multiplicity of a sequencing error in a region with a certain coverage (e.g. a Poisson distribution) and then select a context-specific threshold for the detection based on a certain cut-off in that distribution. Last but not least one could consider the updating

of the k-mer counts once a correction is applied. This would allow for a better statistic of the neighborhood towards following sequencing reads. However, it would also create a dependency on the order of reads and make parallelization more difficult.

3.4.3. Perspective on long reads

At the time of writing, Illumina sequencing is still the major player in the sequencing market [Levy and Myers, 2016]. However, the long read technologies of Oxford Nanopore Technologies and Pacific Bioscience (described in section 1.2) experience continuous improvement. In principle, there is no limitation of the read length for CoCo's approach, but it would greatly suffer from the still very high error rate of this data ($\sim$3-15%). I expect to only rarely find k-mers which do not overlap any error position, and those to hamper the applicability to detect sequencing errors as discrepancies in sample-wise k-mer frequencies. Moreover, CoCo's approach is also not ideally suited to handle indel errors, the dominating type in long read sequencing [Dohm et al., 2020; Goodwin et al., 2016; Van Dijk et al., 2018; Zhang et al., 2020]. These error types can never be masked by non-informative positions, canceling CoCo's advantage to deal with errors in close proximity through spaced k-mers. Despite the growing interest in long read sequencing, I argue that it is still worth investigating to improve upon existing methods for short sequencing read corrections due to several reasons. First of all, I expect Illumina sequencing will still be broadly used in the next decade, as it is cheaper and more accurate. Secondly, short sequencing reads are commonly used together with long reads to compensate for their high error rate [Warwick-Dugdale et al., 2019; Wick et al., 2017; Zhang et al., 2020], and those experiments would greatly benefit from more accurate short reads. Thirdly, there is already a lot of short read sequencing data available in public databases, which requires for accurate error correction tools to distinguish artificial from true diversity, especially in metagenomic samples, to explore their full potential in the downstream analysis.

3.5. Conclusion

Taken all together, CoCo is capable of correcting many sequencing errors, even in close proximity, whereas introducing only very few new errors. This results in very high precision and a drastic reduction of the number of reads with errors. Subsequent assembly with the PenguiN assembler substantially benefits from this much cleaner input. First results of CoCo also in comparison with existing error correction tools suggest that there is potential to outperform them if multiple closely related genomes are present. This highlights CoCo's usability also for mixed genome datatsets and hint that CoCo might also be able to outperform these tools on larger and more complex metagenomic datasets. To confirm this properly, further benchmarks are necessary. However, I could already show that CoCo can process large and complex metagenomic datasets within the PenguiN benchmark. I think CoCo is well suited for this task for four reasons: (1) the count profile allows for a context-considering correction, which seems to be ideal for

a metagenomic, mixture of genomes with highly varying abundances, (2) the use of spaced k-mers allows for accurately detecting errors in close proximity and in low coverage regions, (3) the conservative strategy to apply a correction avoids over-correction, minimizing false-positives and decreasing the risk to switch between strains, which would otherwise lead to a loss of diversity and chimeric assemblies and (4) the lookup table which allows for the efficient storage of the (spaced) k-mer counts, making CoCo applicable also on larger and more complex samples.

In conclusion, CoCo is a reliable method that I believe to scale well to large and complex datasets and ideally interplays with the PenguiN assembler.

4. Concluding remarks

In this thesis, I introduced two new software tools, addressing the challenges of (1) de novo assembly and (2) sequencing error correction in the computational analysis of viral metagenomic samples. There are numerous hints that we are just starting to understand the enormous extent of virus diversity – especially phage diversity – and that there is huge potential in the data that has been previously overlooked and still remains to be discovered. I hope that with the tools and ideas provided in this thesis, I can contribute in helping the community to make new and exciting discoveries by spotting light into the viral dark matter.

References

Ackermann, H.-W. and Prangishvili, D. (2012). Prokaryote viruses studied by electron microscopy. *Archives of virology*, 157(10):1843–1849.

Adriaenssens, E. M., Van Zyl, L., De Maayer, P., Rubagotti, E., Rybicki, E., Tuffin, M., and Cowan, D. A. (2015). Metagenomic analysis of the viral community in n amib d esert hypoliths. *Environmental microbiology*, 17(2):480–495.

Aevarsson, A., Kaczorowska, A.-K., Adalsteinsson, B. T., Ahlqvist, J., Al-Karadaghi, S., Altenbuchner, J., Arsin, H., Átlasson, Ú. Á., Brandt, D., Cichowicz-Cieślak, M., et al. (2021). Going to extremes–a metagenomic journey into the dark matter of life. *FEMS microbiology letters*, 368(12):fnab067.

Akhter, S., Aziz, R. K., and Edwards, R. A. (2012). Phispy: a novel algorithm for finding prophages in bacterial genomes that combines similarity-and composition-based strategies. *Nucleic acids research*, 40(16):e126–e126.

Albright, M. B., Gallegos-Graves, L. V., Feeser, K. L., Montoya, K., Emerson, J. B., Shakya, M., and Dunbar, J. (2022). Experimental evidence for the impact of soil viruses on carbon cycling during surface plant litter decomposition. *ISME Communications*, 2(1):1–8.

Alic, A. S., Ruzafa, D., Dopazo, J., and Blanquer, I. (2016). Objective review of de novo standalone error correction methods for ngs data. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 6(2):111–146.

Amarasinghe, S. L., Su, S., Dong, X., Zappia, L., Ritchie, M. E., and Gouil, Q. (2020). Opportunities and challenges in long-read sequencing data analysis. *Genome biology*, 21(1):1–16.

Antipov, D., Raiko, M., Lapidus, A., and Pevzner, P. A. (2020). Metaviral spades: assembly of viruses from metagenomic data. *Bioinformatics*, 36(14):4126–4129.

Athanasopoulou, K., Boti, M. A., Adamopoulos, P. G., Skourou, P. C., and Scorilas, A. (2021). Third-generation sequencing: The spearhead towards the radical transformation of modern genomics. *Life*, 12(1):30.

Ayling, M., Clark, M. D., and Leggett, R. M. (2020). New approaches for metagenome assembly with short reads. *Briefings in bioinformatics*, 21(2):584–594.

Baaijens, J. A., El Aabidine, A. Z., Rivals, E., and Schönhuth, A. (2017). De novo assembly of viral quasispecies using overlap graphs. *Genome research*, 27(5):835–848.

Baltimore, D. (1971). Expression of animal virus genomes. *Bacteriological reviews*, 35(3):235–241.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. (2012). Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477.

Bbosa, N., Kaleebu, P., and Ssemwanga, D. (2019). Hiv subtype diversity worldwide. *Current Opinion in HIV and AIDS*, 14(3):153–160.

Bell, G. (2021). Evolutionary dynamics of a virus in a vaccinated population. *medRxiv*.

Benoit, G., Lavenier, D., Lemaitre, C., and Rizk, G. (2014). Bloocoo, a memory efficient read corrector. In *European conference on computational biology (ECCB)*.

Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., et al. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *nature*, 456(7218):53–59.

Bergh, Ø., Børsheim, K. Y., Bratbak, G., and Heldal, M. (1989). High abundance of viruses found in aquatic environments. *Nature*, 340(6233):467–468.

Bikel, S., Valdez-Lara, A., Cornejo-Granados, F., Rico, K., Canizales-Quinteros, S., Soberón, X., Del Pozo-Yauner, L., and Ochoa-Leyva, A. (2015). Combining metagenomics, metatranscriptomics and viromics to explore novel microbial interactions: towards a systems-level understanding of human microbiome. *Computational and structural biotechnology journal*, 13:390–401.

Bin Jang, H., Bolduc, B., Zablocki, O., Kuhn, J. H., Roux, S., Adriaenssens, E. M., Brister, J. R., Kropinski, A. M., Krupovic, M., Lavigne, R., et al. (2019). Taxonomic assignment of uncultivated prokaryotic virus genomes is enabled by gene-sharing networks. *Nature biotechnology*, 37(6):632–639.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Boisvert, S., Raymond, F., Godzaridis, É., Laviolette, F., and Corbeil, J. (2012). Ray meta: scalable de novo metagenome assembly and profiling. *Genome biology*, 13(12):1–13.

Bolger, A. M., Lohse, M., and Usadel, B. (2014). Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, 30(15):2114–2120.

Boucher, Y., Douady, C. J., Papke, R. T., Walsh, D. A., Boudreau, M. E. R., Nesbø, C. L., Case, R. J., and Doolittle, W. F. (2003). Lateral gene transfer and the origins of prokaryotic groups. *Annual review of genetics*, 37(1):283–328.

Breitbart, M., Delwart, E., Rosario, K., Segalés, J., Varsani, A., and Consortium, I. R. (2017). Ictv virus taxonomy profile: Circoviridae. *The Journal of general virology*, 98(8):1997.

Breitbart, M., Hewson, I., Felts, B., Mahaffy, J. M., Nulton, J., Salamon, P., and Rohwer, F. (2003). Metagenomic analyses of an uncultured viral community from human feces. *Journal of bacteriology*, 185(20):6220–6223.

Breitbart, M., Miyake, J. H., and Rohwer, F. (2004). Global distribution of nearly identical phage-encoded dna sequences. *FEMS microbiology letters*, 236(2):249–256.

Breitbart, M. and Rohwer, F. (2005). Here a virus, there a virus, everywhere the same virus? *Trends in microbiology*, 13(6):278–284.

Breitbart, M., Salamon, P., Andresen, B., Mahaffy, J. M., Segall, A. M., Mead, D., Azam, F., and Rohwer, F. (2002). Genomic analysis of uncultured marine viral communities. *Proceedings of the National Academy of Sciences*, 99(22):14250–14255.

Breitbart, M., Thompson, L. R., Suttle, C. A., and Sullivan, M. B. (2007). Exploring the vast diversity of marine viruses. *Oceanography*, 20(2):135–139.

Breitwieser, F. P., Lu, J., and Salzberg, S. L. (2019). A review of methods and databases for metagenomic classification and assembly. *Briefings in bioinformatics*, 20(4):1125–1136.

Brown, C. T., Moritz, D., O'Brien, M. P., Reidl, F., Reiter, T., and Sullivan, B. D. (2020). Exploring neighborhoods in large metagenome assembly graphs using spacegraphcats reveals hidden sequence diversity. *Genome biology*, 21(1):1–16.

Brum, J. R. and Sullivan, M. B. (2015). Rising to the challenge: accelerated pace of discovery transforms marine virology. *Nature Reviews Microbiology*, 13(3):147–159.

Bushmanova, E., Antipov, D., Lapidus, A., and Prjibelski, A. D. (2019). rnaspades: a de novo transcriptome assembler and its application to rna-seq data. *GigaScience*, 8(9):giz100.

Bushnell, B. (2014). Bbmap: a fast, accurate, splice-aware aligner. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Callanan, J., Stockdale, S. R., Shkoporov, A., Draper, L. A., Ross, R. P., and Hill, C. (2020). Expansion of known ssrna phage genomes: from tens to over a thousand. *Science advances*, 6(6):eaay5981.

Canzar, S., Andreotti, S., Weese, D., Reinert, K., and Klau, G. W. (2016). Cidane: comprehensive isoform discovery and abundance estimation. *Genome biology*, 17(1):1–18.

Cepeda, V., Liu, B., Almeida, M., Hill, C. M., Koren, S., Treangen, T. J., and Pop, M. (2017). Metacompass: reference-guided assembly of metagenomes. *BioRxiv*, page 212506.

Chaisson, M. J., Brinza, D., and Pevzner, P. A. (2009). De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome research*, 19(2):336–346.

Chaitanya, K. (2019). Structure and organization of virus genomes. In *Genome and Genomics*, pages 1–30. Springer.

Chamakura, K. R., Tran, J. S., O'Leary, C., Lisciandro, H. G., Antillon, S. F., Garza, K. D., Tran, E., Min, L., and Young, R. (2020). Rapid de novo evolution of lysis genes in single-stranded rna phages. *Nature communications*, 11(1):1–11.

Check Hayden, E. (2014). Is the $1,000 genome for real? *Nature*.

Chen, J., Zhao, Y., and Sun, Y. (2018). De novo haplotype reconstruction in viral quasispecies using paired-end read guided path finding. *Bioinformatics*, 34(17):2927–2935.

Chikhi, R. and Medvedev, P. (2014). Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37.

Clark, S. J., Lee, H. J., Smallwood, S. A., Kelsey, G., and Reik, W. (2016). Single-cell epigenomics: powerful new methods for understanding gene regulation and cell identity. *Genome biology*, 17(1):1–10.

Clarke, J., Wu, H.-C., Jayasinghe, L., Patel, A., Reid, S., and Bayley, H. (2009). Continuous base identification for single-molecule nanopore dna sequencing. *Nature nanotechnology*, 4(4):265–270.

Colavecchio, A., Cadieux, B., Lo, A., and Goodridge, L. D. (2017). Bacteriophages contribute to the spread of antibiotic resistance genes among foodborne pathogens of the enterobacteriaceae family–a review. *Frontiers in Microbiology*, 8:1108.

Compeau, P. E., Pevzner, P. A., and Tesler, G. (2011a). How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991.

Compeau, P. E., Pevzner, P. A., and Tesler, G. (2011b). Why are de bruijn graphs useful for genome assembly? *Nature biotechnology*, 29(11):987.

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., Szcześniak, M. W., Gaffney, D. J., Elo, L. L., Zhang, X., et al. (2016). A survey of best practices for rna-seq data analysis. *Genome biology*, 17(1):1–19.

Coutinho, F. H., Silveira, C. B., Gregoracci, G. B., Thompson, C. C., Edwards, R. A., Brussaard, C. P., Dutilh, B. E., and Thompson, F. L. (2017). Marine viruses discovered via metagenomics shed light on viral strategies throughout the oceans. *Nature communications*, 8(1):1–12.

Craigie, R. and Bushman, F. D. (2012). Hiv dna integration. *Cold Spring Harbor perspectives in medicine*, 2(7):a006890.

Cudini, J., Roy, S., Houldcroft, C. J., Bryant, J. M., Depledge, D. P., Tutill, H., Veys, P., Williams, R., Worth, A. J., Tamuri, A. U., et al. (2019). Human cytomegalovirus haplotype reconstruction reveals high diversity due to superinfection and evidence of within-host recombination. *Proceedings of the National Academy of Sciences*, 116(12):5693–5698.

Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55.

Danovaro, R., Corinaldesi, C., Dell'Anno, A., Fuhrman, J. A., Middelburg, J. J., Noble, R. T., and Suttle, C. A. (2011). Marine viruses and global climate change. *FEMS microbiology reviews*, 35(6):993–1034.

Dávila-Ramos, S., Castelán-Sánchez, H. G., Martínez-Ávila, L., Sánchez-Carbente, M. d. R., Peralta, R., Hernández-Mendoza, A., Dobson, A. D., Gonzalez, R. A., Pastor, N., and Batista-García, R. A. (2019). A review on viral metagenomics in extreme environments. *Frontiers in microbiology*, page 2403.

Deng, Z. and Delwart, E. (2021). Contigextender: a new approach to improving de novo sequence assembly for viral metagenomics data. *BMC bioinformatics*, 22(1):1–19.

Deng, Z.-L., Dhingra, A., Fritz, A., Götting, J., Münch, P. C., Steinbrück, L., Schulz, T. F., Ganzenmüller, T., and McHardy, A. C. (2021). Evaluating assembly and variant calling software for strain-resolved analysis of large dna viruses. *Briefings in bioinformatics*, 22(3):bbaa123.

Desai, N., Antonopoulos, D., Gilbert, J. A., Glass, E. M., and Meyer, F. (2012). From genomics to metagenomics. *Current opinion in biotechnology*, 23(1):72–76.

Désiré, N., Cerutti, L., Le Hingrat, Q., Perrier, M., Emler, S., Calvez, V., Descamps, D., Marcelin, A.-G., Hué, S., and Visseaux, B. (2018). Characterization update of hiv-1 m subtypes diversity and proposal for subtypes a and d sub-subtypes reclassification. *Retrovirology*, 15(1):1–7.

Dion, M. B., Oechslin, F., and Moineau, S. (2020). Phage diversity, genomics and phylogeny. *Nature Reviews Microbiology*, 18(3):125–138.

Dohm, J. C., Peters, P., Stralis-Pavese, N., and Himmelbauer, H. (2020). Benchmarking of long-read correction methods. *NAR Genomics and Bioinformatics*, 2(2):lqaa037.

Dolan, P. T., Whitfield, Z. J., and Andino, R. (2018). Mechanisms and concepts in rna virus population dynamics and evolution. *Annual Review of Virology*, 5:69–92.

Dorigo, U., Jacquet, S., and Humbert, J.-F. (2004). Cyanophage diversity, inferred from g20 gene analyses, in the largest natural lake in france, lake bourget. *Applied and Environmental Microbiology*, 70(2):1017–1022.

Drezen, E., Rizk, G., Chikhi, R., Deltel, C., Lemaitre, C., Peterlongo, P., and Lavenier, D. (2014). Gatb: genome assembly & analysis tool box. *Bioinformatics*, 30(20):2959–2961.

Eddy, S. R. et al. (1995). Multiple alignment using hidden markov models. In *Ismb*, volume 3, pages 114–120.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., et al. (2009). Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138.

Emerson, J. B., Roux, S., Brum, J. R., Bolduc, B., Woodcroft, B. J., Jang, H. B., Singleton, C. M., Solden, L. M., Naas, A. E., Boyd, J. A., et al. (2018). Host-linked soil viral ecology along a permafrost thaw gradient. *Nature microbiology*, 3(8):870–880.

Erbert, M., Rechner, S., and Müller-Hannemann, M. (2017). Gerbil: a fast and memory-efficient k-mer counter with gpu-support. *Algorithms for Molecular Biology*, 12(1):1–12.

Filée, J., Tétart, F., Suttle, C. A., and Krisch, H. (2005). Marine t4-type bacteriophages, a ubiquitous component of the dark matter of the biosphere. *Proceedings of the National Academy of Sciences*, 102(35):12471–12476.

Fox, E. J., Reid-Bayliss, K. S., Emond, M. J., and Loeb, L. A. (2014). Accuracy of next generation sequencing platforms. *Next generation, sequencing & applications*, 1.

Frazão, N., Sousa, A., Lässig, M., and Gordo, I. (2019). Horizontal gene transfer overrides mutation in escherichia coli colonizing the mammalian gut. *Proceedings of the National Academy of Sciences*, 116(36):17906–17915.

Fritz, A., Bremges, A., Deng, Z.-L., Lesker, T. R., Götting, J., Ganzenmueller, T., Sczyrba, A., Dilthey, A., Klawonn, F., and McHardy, A. C. (2021). Haploflow: Strain-resolved de novo assembly of viral genomes. *Genome Biology*, 22(1):1–19.

Fu, L., Niu, B., Zhu, Z., Wu, S., and Li, W. (2012). Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152.

Fuhrman, J. A. (1999). Marine viruses and their biogeochemical and ecological effects. *Nature*, 399(6736):541–548.

Fuller, C. W., Middendorf, L. R., Benner, S. A., Church, G. M., Harris, T., Huang, X., Jovanovich, S. B., Nelson, J. R., Schloss, J. A., Schwartz, D. C., et al. (2009). The challenges of sequencing by synthesis. *Nature biotechnology*, 27(11):1013–1023.

García-López, R., Vázquez-Castellanos, J. F., and Moya, A. (2015). Fragmentation and coverage variation in viral metagenome assemblies, and their effect in diversity calculations. *Frontiers in bioengineering and biotechnology*, 3:141.

Ghurye, J. S., Cepeda-Espinoza, V., and Pop, M. (2016). Focus: microbiome: metagenomic assembly: overview, challenges and applications. *The Yale journal of biology and medicine*, 89(3):353.

Gogarten, J. P. and Townsend, J. P. (2005). Horizontal gene transfer, genome innovation and evolution. *Nature Reviews Microbiology*, 3(9):679–687.

Goodwin, S., McPherson, J. D., and McCombie, W. R. (2016). Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351.

Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., Adiconis, X., Fan, L., Raychowdhury, R., Zeng, Q., et al. (2011). Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, 29(7):644–652.

Green, P. (1996). Phrap documentation. http://www.phrap.org/phredphrap/phrap.html. Accessed 15 May 2021.

Gregory, A. C., Zablocki, O., Zayed, A. A., Howell, A., Bolduc, B., and Sullivan, M. B. (2020). The gut virome database reveals age-dependent patterns of virome diversity in the human gut. *Cell host & microbe*, 28(5):724–740.

Gregory, A. C., Zayed, A. A., Conceição-Neto, N., Temperton, B., Bolduc, B., Alberti, A., Ardyna, M., Arkhipova, K., Carmichael, M., Cruaud, C., et al. (2019). Marine dna viral macro-and microdiversity from pole to pole. *Cell*, 177(5):1109–1123.

Guerin, E. and Hill, C. (2020). Shining light on human gut bacteriophages. *Frontiers in cellular and infection microbiology*, page 481.

Gulyaeva, A., Garmaeva, S., Ruigrok, R. A., Wang, D., Riksen, N. P., Netea, M. G., Wijmenga, C., Weersma, R. K., Fu, J., Vila, A. V., et al. (2022). Discovery, diversity, and functional associations of crass-like phages in human gut metagenomes from four dutch cohorts. *Cell reports*, 38(2):110204.

Guo, J., Bolduc, B., Zayed, A. A., Varsani, A., Dominguez-Huerta, G., Delmont, T. O., Pratama, A. A., Gazitúa, M. C., Vik, D., Sullivan, M. B., et al. (2021). Virsorter2: a multi-classifier, expert-guided approach to detect diverse dna and rna viruses. *Microbiome*, 9(1):1–13.

Gyles, C. and Boerlin, P. (2014). Horizontally transferred genetic elements and their role in pathogenesis of bacterial disease. *Veterinary pathology*, 51(2):328–340.

Gytz, H., Mohr, D., Seweryn, P., Yoshimura, Y., Kutlubaeva, Z., Dolman, F., Chelchessa, B., Chetverin, A. B., Mulder, F. A., Brodersen, D. E., et al. (2015). Structural basis for rna-genome recognition during bacteriophage qβ replication. *Nucleic acids research*, 43(22):10893–10906.

Haider, B., Ahn, T.-H., Bushnell, B., Chai, J., Copeland, A., and Pan, C. (2014). Omega: an overlap-graph de novo assembler for metagenomics. *Bioinformatics*, 30(19):2717–2722.

Handelsman, J. (2004). Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and molecular biology reviews*, 68(4):669–685.

Handelsman, J., Rondon, M. R., Brady, S. F., Clardy, J., and Goodman, R. M. (1998). Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & biology*, 5(10):R245–R249.

Hauser, M., Steinegger, M., and Söding, J. (2016). Mmseqs software suite for fast and deep clustering and searching of large protein sequence sets. *Bioinformatics*, 32(9):1323–1330.

Hayes, S., Mahony, J., Nauta, A., and Van Sinderen, D. (2017). Metagenomic approaches to assess bacteriophages in various environmental niches. *Viruses*, 9(6):127.

Heather, J. M. and Chain, B. (2016). The sequence of sequencers: The history of sequencing dna. *Genomics*, 107(1):1–8.

Hemelaar, J., Gouws, E., Ghys, P. D., and Osmanov, S. (2006). Global and regional distribution of hiv-1 genetic subtypes and recombinants in 2004. *Aids*, 20(16):W13–W23.

Heo, Y., Wu, X.-L., Chen, D., Ma, J., and Hwu, W.-M. (2014). Bless: bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, 30(10):1354–1362.

Heydari, M., Miclotte, G., Demeester, P., Van de Peer, Y., and Fostier, J. (2017). Evaluation of the impact of illumina error correction tools on de novo genome assembly. *BMC bioinformatics*, 18(1):1–13.

Hölzer, M. and Marz, M. (2019). De novo transcriptome assembly: A comprehensive cross-species comparison of short-read rna-seq assemblers. *Gigascience*, 8(5):giz039.

Hopkins, M., Kailasan, S., Cohen, A., Roux, S., Tucker, K. P., Shevenell, A., Agbandje-McKenna, M., and Breitbart, M. (2014). Diversity of environmental single-stranded dna phages revealed by pcr amplification of the partial major capsid protein. *The ISME journal*, 8(10):2093–2103.

Hugenholtz, P., Goebel, B. M., and Pace, N. R. (1998). Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *Journal of bacteriology*, 180(18):4765–4774.

Hugenholtz, P. and Pace, N. R. (1996). Identifying microbial diversity in the natural environment: a molecular phylogenetic approach. *Trends in biotechnology*, 14(6):190–197.

Hunkapiller, T., Kaiser, R., Koop, B., and Hood, L. (1991). Large-scale and automated dna sequence determination. *Science*, 254(5028):59–67.

Hunt, M., Gall, A., Ong, S. H., Brener, J., Ferns, B., Goulder, P., Nastouli, E., Keane, J. A., Kellam, P., and Otto, T. D. (2015). Iva: accurate de novo assembly of rna virus genomes. *Bioinformatics*, 31(14):2374–2376.

Hunt, M., Newbold, C., Berriman, M., and Otto, T. D. (2014). A comprehensive evaluation of assembly scaffolding tools. *Genome biology*, 15(3):1–15.

Huson, D. H., Reinert, K., and Myers, E. W. (2002). The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM (JACM)*, 49(5):603–615.

Hutchison, C. A., Smith, H. O., Pfannkoch, C., and Venter, J. C. (2005). Cell-free cloning using φ29 dna polymerase. *Proceedings of the National Academy of Sciences*, 102(48):17332–17336.

Hwang, J., Park, S. Y., Park, M., Lee, S., and Lee, T.-K. (2017). Seasonal dynamics and metagenomic characterization of marine viruses in goseong bay, korea. *PloS one*, 12(1):e0169841.

Hyatt, D., Chen, G.-L., LoCascio, P. F., Land, M. L., Larimer, F. W., and Hauser, L. J. (2010). Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC bioinformatics*, 11(1):1–11.

Ihrmark, K., Bödeker, I., Cruz-Martinez, K., Friberg, H., Kubartova, A., Schenck, J., Strid, Y., Stenlid, J., Brandström-Durling, M., Clemmensen, K. E., et al. (2012). New primers to amplify the fungal its2 region–evaluation by 454-sequencing of artificial and natural communities. *FEMS microbiology ecology*, 82(3):666–677.

Ilie, L., Fazayeli, F., and Ilie, S. (2011). Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302.

International Committee on Taxonomy of Viruses Executive Committee (2020). The new scope of virus taxonomy: partitioning the virosphere into 15 hierarchical ranks. *Nature Microbiology*, 5(5):668.

Irwin, N. A., Pittis, A. A., Richards, T. A., and Keeling, P. J. (2022). Systematic evaluation of horizontal gene transfer between eukaryotes and viruses. *Nature microbiology*, 7(2):327–336.

Jain, C., Rodriguez-R, L. M., Phillippy, A. M., Konstantinidis, K. T., and Aluru, S. (2018). High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries. *Nature communications*, 9(1):1–8.

Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., Dangl, J. L., and Jones, C. D. (2007). Extending assembly of short dna sequences to handle error. *Bioinformatics*, 23(21):2942–2944.

Jensen, T. Z., Niemann, J., Iversen, K. H., Fotakis, A. K., Gopalakrishnan, S., Vågene, Å. J., Pedersen, M. W., Sinding, M.-H. S., Ellegaard, M. R., Allentoft, M. E., et al. (2019). A 5700 year-old human genome and oral microbiome from chewed birch pitch. *Nature Communications*, 10(1):1–10.

Johansen, J., Plichta, D. R., Nissen, J. N., Jespersen, M. L., Shah, S. A., Deng, L., Stokholm, J., Bisgaard, H., Nielsen, D. S., Sørensen, S. J., et al. (2022). Genome binning of viral entities from bulk metagenomics data. *Nature communications*, 13(1):1–12.

Johnson, J. S., Spakowicz, D. J., Hong, B.-Y., Petersen, L. M., Demkowicz, P., Chen, L., Leopold, S. R., Hanson, B. M., Agresta, H. O., Gerstein, M., et al. (2019). Evaluation of 16s rrna gene sequencing for species and strain-level microbiome analysis. *Nature communications*, 10(1):1–11.

Jouffret, V., Miotello, G., Culotta, K., Ayrault, S., Pible, O., and Armengaud, J. (2021). Increasing the power of interpretation for soil metaproteomics data. *Microbiome*, 9(1):1–15.

Kang, D. D., Froula, J., Egan, R., and Wang, Z. (2015). Metabat, an efficient tool for accurately reconstructing single genomes from complex microbial communities. *PeerJ*, 3:e1165.

Kao, W.-C., Chan, A. H., and Song, Y. S. (2011). Echo: a reference-free short-read error correction algorithm. *Genome research*, 21(7):1181–1192.

Katoh, K., Misawa, K., Kuma, K.-i., and Miyata, T. (2002). Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, 30(14):3059–3066.

Kchouk, M., Gibrat, J.-F., and Elloumi, M. (2017). Generations of sequencing technologies: from first to next generation. *Biology and Medicine*, 9(3).

Keen, E. C. (2015). A century of phage research: bacteriophages and the shaping of modern biology. *Bioessays*, 37(1):6–9.

Kelley, D. R., Schatz, M. C., and Salzberg, S. L. (2010). Quake: quality-aware detection and correction of sequencing errors. *Genome biology*, 11(11):1–13.

Kieft, K., Zhou, Z., and Anantharaman, K. (2020). Vibrant: automated recovery, annotation and curation of microbial viruses, and evaluation of viral community function from genomic sequences. *Microbiome*, 8(1):1–23.

Kim, P.-G., Cho, H.-G., and Park, K. (2008). A scaffold analysis tool using mate-pair information in genome sequencing. *Journal of Biomedicine and Biotechnology*.

Kleiner, M., Hooper, L. V., and Duerkop, B. A. (2015). Evaluation of methods to purify virus-like particles for metagenomic sequencing of intestinal viromes. *BMC genomics*, 16(1):1–15.

Kokot, M., Długosz, M., and Deorowicz, S. (2017). Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761.

Koonin, E. V., Krupovic, M., and Agol, V. I. (2021). The baltimore classification of viruses 50 years later: How does it stand in the light of virus evolution? *Microbiology and Molecular Biology Reviews*, 85(3):e00053–21.

Kounosu, A., Murase, K., Yoshida, A., Maruyama, H., and Kikuchi, T. (2019). Improved 18s and 28s rdna primer sets for ngs-based parasite detection. *Scientific reports*, 9(1):1–12.

Kraft, L. (2021). Adapting a novel metagenome assembler for transcriptomic data: A proof of concept study. *Unpublished master thesis*.

Krishnamurthy, S. R., Janowski, A. B., Zhao, G., Barouch, D., and Wang, D. (2016). Hyperexpansion of rna bacteriophage diversity. *PLoS biology*, 14(3):e1002409.

Langille, M. G., Zaneveld, J., Caporaso, J. G., McDonald, D., Knights, D., Reyes, J. A., Clemente, J. C., Burkepile, D. E., Thurber, R. L. V., Knight, R., et al. (2013). Predictive functional profiling of microbial communities using 16s rrna marker gene sequences. *Nature biotechnology*, 31(9):814–821.

Lapidus, A. L. and Korobeynikov, A. I. (2021). Metagenomic data assembly–the way of decoding unknown microorganisms. *Frontiers in Microbiology*, 12:653.

Laserson, J., Jojic, V., and Koller, D. (2011). Genovo: de novo assembly for metagenomes. *Journal of Computational Biology*, 18(3):429–443.

Lefkowitz, E. J., Dempsey, D. M., Hendrickson, R. C., Orton, R. J., Siddell, S. G., and Smith, D. B. (2018). Virus taxonomy: the database of the international committee on taxonomy of viruses (ictv). *Nucleic acids research*, 46(D1):D708–D717.

Levy, S. E. and Myers, R. M. (2016). Advancements in next-generation sequencing. *Annual review of genomics and human genetics*, 17:95–115.

Leye, N., Vidal, N., Ndiaye, O., Diop-Ndiaye, H., Wade, A. S., Mboup, S., Delaporte, E., Toure-Kane, C., and Peeters, M. (2013). High frequency of hiv-1 infections with multiple hiv-1 strains in men having sex with men (msm) in senegal. *Infection, Genetics and Evolution*, 20:206–214.

Li, D., Liu, C.-M., Luo, R., Sadakane, K., and Lam, T.-W. (2015). Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676.

Li, H. (2009). kseq. http://lh3lh3.users.sourceforge.net/kseq.shtml. Accessed 18 November 2020.

Li, H. (2011). Wgsim. https://github.com/lh3/wgsim. Accessed 15 March 2021.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*.

Li, H. (2015). Bfc: correcting illumina sequencing errors. *Bioinformatics*, 31(17):2885–2887.

Li, Z., Chen, Y., Mu, D., Yuan, J., Shi, Y., Zhang, H., Gan, J., Li, N., Hu, X., Liu, B., et al. (2012). Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1):25–37.

Liang, G. and Bushman, F. D. (2021). The human virome: assembly, composition and host interactions. *Nature Reviews Microbiology*, 19(8):514–527.

Liao, X., Li, M., Zou, Y., Wu, F.-X., Wang, J., et al. (2019). Current challenges and solutions of de novo assembly. *Quantitative Biology*, 7(2):90–109.

Limasset, A., Flot, J.-F., and Peterlongo, P. (2020). Toward perfect reads: self-correction of short reads via mapping on de bruijn graphs. *Bioinformatics*, 36(5):1374–1381.

Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., Lin, D., Lu, L., and Law, M. (2012). Comparison of next-generation sequencing systems. *Journal of Biomedicine and Biotechnology*, 2012.

Liu, Y., Schröder, J., and Schmidt, B. (2013). Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315.

Lodish, H. F. (1968). Bacteriophage f2 rna: control of translation and gene order. *Nature*, 220(5165):345–350.

Louten, J. (2016). Virus structure and classification. *Essential Human Virology*, page 19.

Lozupone, C. A. and Knight, R. (2007). Global patterns in bacterial diversity. *Proceedings of the National Academy of Sciences*, 104(27):11436–11440.

Lynch, M. (2010). Evolution of the mutation rate. *TRENDS in Genetics*, 26(8):345–352.

Ma, X., Shao, Y., Tian, L., Flasch, D. A., Mulder, H. L., Edmonson, M. N., Liu, Y., Chen, X., Newman, S., Nakitandwe, J., et al. (2019). Analysis of error profiles in deep next-generation sequencing data. *Genome biology*, 20(1):1–15.

Magoč, T. and Salzberg, S. L. (2011). Flash: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, 27(21):2957–2963.

Maldarelli, F. et al. (2016). The role of hiv integration in viral persistence: no more whistling past the proviral graveyard. *The Journal of clinical investigation*, 126(2):438–447.

Mamun, A.-A., Pal, S., and Rajasekaran, S. (2016). Kcmbt: ak-mer counter based on multiple burst trees. *Bioinformatics*, 32(18):2783–2790.

Manrao, E. A., Derrington, I. M., Laszlo, A. H., Langford, K. W., Hopper, M. K., Gillgren, N., Pavlenok, M., Niederweis, M., and Gundlach, J. H. (2012). Reading dna at single-nucleotide resolution with a mutant mspa nanopore and phi29 dna polymerase. *Nature biotechnology*, 30(4):349–353.

Manrique, P., Bolduc, B., Walk, S. T., van der Oost, J., de Vos, W. M., and Young, M. J. (2016). Healthy human gut phageome. *Proceedings of the National Academy of Sciences*, 113(37):10400–10405.

Marçais, G. and Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., et al. (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380.

Marinier, E., Brown, D. G., and McConkey, B. J. (2015). Pollux: platform independent error correction of single and mixed genomes. *BMC bioinformatics*, 16(1):1–12.

Marintcheva, B. (2018). *"Chapter 1 - Introduction to Viral Structure, Diversity and Biology." Harnessing the Power of Viruses*. Academic Press.

Martin, J. A. and Wang, Z. (2011). Next-generation transcriptome assembly. *Nature Reviews Genetics*, 12(10):671–682.

Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet. journal*, 17(1):10–12.

Matsuo, Y., Komiya, S., Yasumizu, Y., Yasuoka, Y., Mizushima, K., Takagi, T., Kryukov, K., Fukuda, A., Morimoto, Y., Naito, Y., et al. (2021). Full-length 16s rrna gene amplicon analysis

of human gut microbiota using minion™ nanopore sequencing confers species-level resolution. *BMC microbiology*, 21(1):1–13.

Maxam, A. M. and Gilbert, W. (1977). A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564.

Maxam, A. M. and Gilbert, W. (1980). [57] sequencing end-labeled dna with base-specific chemical cleavages. In *Methods in enzymology*, volume 65, pages 499–560. Elsevier.

McNair, K., Aziz, R. K., Pusch, G. D., Overbeek, R., Dutilh, B. E., and Edwards, R. (2018). Phage genome annotation using the rast pipeline. In *Bacteriophages*, pages 231–238. Springer.

Medvedev, P., Georgiou, K., Myers, G., and Brudno, M. (2007). Computability of models for sequence assembly. In *International Workshop on Algorithms in Bioinformatics*, pages 289–301. Springer.

Meleshko, D., Hajirasouliha, I., and Korobeynikov, A. (2022). coronaspades: from biosynthetic gene clusters to rna viral assemblies. *Bioinformatics*, 38(1):1–8.

Meyer, F., Fritz, A., Deng, Z.-L., Koslicki, D., Gurevich, A., Robertson, G., Alser, M., Antipov, D., Beghini, F., Bertrand, D., et al. (2021). Critical assessment of metagenome interpretation-the second round of challenges. *BioRxiv*.

Mihara, T., Nishimura, Y., Shimizu, Y., Nishiyama, H., Yoshikawa, G., Uehara, H., Hingamp, P., Goto, S., and Ogata, H. (2016). Linking virus genomes with host taxonomy. *Viruses*, 8(3):66.

Mikheenko, A., Saveliev, V., and Gurevich, A. (2016a). Metaquast: evaluation of metagenome assemblies. *Bioinformatics*, 32(7):1088–1090.

Mikheenko, A., Valin, G., Prjibelski, A., Saveliev, V., and Gurevich, A. (2016b). Icarus: visualizer for de novo assembly evaluation. *Bioinformatics*, 32(21):3321–3323.

Miller, J. R., Koren, S., and Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327.

Miller, R. R., Montoya, V., Gardy, J. L., Patrick, D. M., and Tang, P. (2013). Metagenomics for pathogen detection in public health. *Genome medicine*, 5(9):1–14.

Minot, S., Bryson, A., Chehoud, C., Wu, G. D., Lewis, J. D., and Bushman, F. D. (2013). Rapid evolution of the human gut virome. *Proceedings of the National Academy of Sciences*, 110(30):12450–12455.

Mirzaei, M. K. and Maurice, C. F. (2017). Ménage à trois in the human gut: interactions between host, bacteria and phages. *Nature Reviews Microbiology*, 15(7):397–408.

Mirzaei, M. K., Xue, J., Costa, R., Ru, J., Schulz, S., Taranu, Z. E., and Deng, L. (2021). Challenges of studying the human virome–relevant emerging technologies. *Trends in Microbiology*, 29(2):171–181.

Mitchell, K., Brito, J. J., Mandric, I., Wu, Q., Knyazev, S., Chang, S., Martin, L. S., Karlsberg, A., Gerasimov, E., Littman, R., et al. (2020). Benchmarking of computational error-correction methods for next-generation sequencing data. *Genome biology*, 21(1):1–13.

Mokili, J. L., Rohwer, F., and Dutilh, B. E. (2012). Metagenomics and future perspectives in virus discovery. *Current opinion in virology*, 2(1):63–77.

Moon, K., Jeon, J. H., Kang, I., Park, K. S., Lee, K., Cha, C.-J., Lee, S. H., and Cho, J.-C. (2020). Freshwater viral metagenome reveals novel and functional phage-borne antibiotic resistance genes. *Microbiome*, 8(1):1–15.

Moreno-Gallego, J. L., Chou, S.-P., Di Rienzi, S. C., Goodrich, J. K., Spector, T. D., Bell, J. T., Youngblut, N. D., Hewson, I., Reyes, A., and Ley, R. E. (2019). Virome diversity correlates with intestinal microbiome diversity in adult monozygotic twins. *Cell host & microbe*, 25(2):261–272.

Myers, E. W. (2005). The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85.

Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H., Remington, K. A., et al. (2000). A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204.

Namiki, T., Hachiya, T., Tanaka, H., and Sakakibara, Y. (2012). Metavelvet: an extension of velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic acids research*, 40(20):e155–e155.

Nayfach, S., Camargo, A. P., Schulz, F., Eloe-Fadrosh, E., Roux, S., and Kyrpides, N. C. (2021a). Checkv assesses the quality and completeness of metagenome-assembled viral genomes. *Nature biotechnology*, 39(5):578–585.

Nayfach, S., Páez-Espino, D., Call, L., Low, S. J., Sberro, H., Ivanova, N. N., Proal, A. D., Fischbach, M. A., Bhatt, A. S., Hugenholtz, P., et al. (2021b). Metagenomic compendium of 189,680 dna viruses from the human gut microbiome. *Nature Microbiology*, 6(7):960–970.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

Nikolenko, S. I., Korobeynikov, A. I., and Alekseyev, M. A. (2013). Bayeshammer: Bayesian clustering for error correction in single-cell sequencing. In *BMC genomics*, volume 14, pages 1–11. Springer.

Norman, J. M., Handley, S. A., Baldridge, M. T., Droit, L., Liu, C. Y., Keller, B. C., Kambal, A., Monaco, C. L., Zhao, G., Fleshner, P., et al. (2015). Disease-specific alterations in the enteric virome in inflammatory bowel disease. *Cell*, 160(3):447–460.

Nurk, S., Meleshko, D., Korobeynikov, A., and Pevzner, P. A. (2017). metaspades: a new versatile metagenomic assembler. *Genome research*, 27(5):824–834.

Nyrén, P., Pettersson, B., and Uhlén, M. (1993). Solid phase dna minisequencing by an enzymatic luminometric inorganic pyrophosphate detection assay. *Analytical biochemistry*, 208(1):171–175.

O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., et al. (2016). Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, 44(D1):D733–D745.

Olsen, G. J., Lane, D. J., Giovannoni, S. J., Pace, N. R., and Stahl, D. A. (1986). Microbial ecology and evolution: a ribosomal rna approach. *Annual reviews in microbiology*, 40(1):337–365.

Ozsolak, F. and Milos, P. M. (2011). Rna sequencing: advances, challenges and opportunities. *Nature reviews genetics*, 12(2):87–98.

Pace, N. R. (1997). A molecular view of microbial diversity and the biosphere. *Science*, 276(5313):734–740.

Pace, N. R., Stahl, D. A., Lane, D. J., and Olsen, G. J. (1986). The analysis of natural microbial populations by ribosomal rna sequences. In *Advances in microbial ecology*, pages 1–55. Springer.

Palermo, C. N., Fulthorpe, R. R., Saati, R., and Short, S. M. (2019). Metagenomic analysis of virus diversity and relative abundance in a eutrophic freshwater harbour. *Viruses*, 11(9):792.

Payne, A., Holmes, N., Rakyan, V., and Loose, M. (2019). Bulkvis: a graphical viewer for oxford nanopore bulk fast5 files. *Bioinformatics*, 35(13):2193–2198.

Peng, Y., Leung, H. C., Yiu, S.-M., and Chin, F. Y. (2012). Idba-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428.

Pereira Andrade, A. C. d. S., Victor de Miranda Boratto, P., Rodrigues, R. A. L., Bastos, T. M., Azevedo, B. L., Dornas, F. P., Oliveira, D. B., Drumond, B. P., Kroon, E. G., and Abrahão, J. S. (2019). New isolates of pandoraviruses: contribution to the study of replication cycle steps. *Journal of virology*, 93(5):e01942–18.

Pereira-Gómez, M. and Sanjuán, R. (2015). Effect of mismatch repair on the mutation rate of bacteriophage ϕx174. *Virus Evolution*, 1(1):vev010.

Pérez-Cobas, A. E., Gomez-Valero, L., and Buchrieser, C. (2020). Metagenomic approaches in microbial ecology: an update on whole-genome and marker gene sequencing analyses. *Microbial genomics*, 6(8).

Pérez-Losada, M., Arenas, M., Galán, J. C., Palero, F., and González-Candelas, F. (2015). Recombination in viruses: mechanisms, methods of study, and evolutionary consequences. *Infection, Genetics and Evolution*, 30:296–307.

Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. *Proceedings of the national academy of sciences*, 98(17):9748–9753.

Pfeiffer, F., Gröber, C., Blank, M., Händler, K., Beyer, M., Schultze, J. L., and Mayer, G. (2018). Systematic evaluation of error rates and causes in short samples in next-generation sequencing. *Scientific reports*, 8(1):1–14.

Philippe, N., Legendre, M., Doutre, G., Couté, Y., Poirot, O., Lescot, M., Arslan, D., Seltzer, V., Bertaux, L., Bruley, C., et al. (2013). Pandoraviruses: amoeba viruses with genomes up to 2.5 mb reaching that of parasitic eukaryotes. *Science*, 341(6143):281–286.

Pinzone, M. R., VanBelzen, D. J., Weissman, S., Bertuccio, M. P., Cannon, L., Venanzi-Rullo, E., Migueles, S., Jones, R. B., Mota, T., Joseph, S. B., et al. (2019). Longitudinal hiv sequencing reveals reservoir expression leading to decay which is obscured by clonal expansion. *Nature communications*, 10(1):1–12.

Pop, M. (2009). Genome assembly reborn: recent computational challenges. *Briefings in bioinformatics*, 10(4):354–366.

Pride, D. T., Meinersmann, R. J., Wassenaar, T. M., and Blaser, M. J. (2003). Evolutionary implications of microbial genome tetranucleotide frequency biases. *Genome research*, 13(2):145–158.

Qu, W., Hashimoto, S.-i., and Morishita, S. (2009). Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome research*, 19(7):1309–1315.

Quince, C., Walker, A. W., Simpson, J. T., Loman, N. J., and Segata, N. (2017). Shotgun metagenomics, from sampling to analysis. *Nature biotechnology*, 35(9):833–844.

Raghavan, V., Kraft, L., Mesny, F., and Rigerte, L. (2022). A simple guide to de novo transcriptome assembly and annotation. *Briefings in bioinformatics*, 23(2):bbab563.

Raoult, D., Audic, S., Robert, C., Abergel, C., Renesto, P., Ogata, H., La Scola, B., Suzan, M., and Claverie, J.-M. (2004). The 1.2-megabase genome sequence of mimivirus. *science*, 306(5700):1344–1350.

Ren, J., Ahlgren, N. A., Lu, Y. Y., Fuhrman, J. A., and Sun, F. (2017). Virfinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*, 5(1):1–20.

Retel, C., Märkle, H., Becks, L., and Feulner, P. G. (2019). Ecological and evolutionary processes shaping viral genetic diversity. *Viruses*, 11(3):220.

Reuter, J. A., Spacek, D. V., and Snyder, M. P. (2015). High-throughput sequencing technologies. *Molecular cell*, 58(4):586–597.

Reyes, A., Haynes, M., Hanson, N., Angly, F. E., Heath, A. C., Rohwer, F., and Gordon, J. I. (2010). Viruses in the faecal microbiota of monozygotic twins and their mothers. *Nature*, 466(7304):334–338.

Rizk, G., Lavenier, D., and Chikhi, R. (2013). Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653.

Rizzi, R., Beretta, S., Patterson, M., Pirola, Y., Previtali, M., Della Vedova, G., and Bonizzoni, P. (2019). Overlap graphs and de bruijn graphs: data structures for de novo genome assembly in the big data era. *Quantitative Biology*, 7(4):278–292.

Rohwer, F., Seguritan, V., Choi, D., Segall, A., and Azam, F. (2001). Production of shotgun libraries using random amplification. *Biotechniques*, 31(1):108–118.

Rose, R., Constantinides, B., Tapinos, A., Robertson, D. L., and Prosperi, M. (2016). Challenges in the analysis of viral metagenomes. *Virus Evolution*, 2(2):vew022.

Roux, S., Adriaenssens, E. M., Dutilh, B. E., Koonin, E. V., Kropinski, A. M., Krupovic, M., Kuhn, J. H., Lavigne, R., Brister, J. R., Varsani, A., et al. (2019a). Minimum information about an uncultivated virus genome (miuvig). *Nature biotechnology*, 37(1):29–37.

Roux, S., Brum, J. R., Dutilh, B. E., Sunagawa, S., Duhaime, M. B., Loy, A., Poulos, B. T., Solonenko, N., Lara, E., Poulain, J., et al. (2016). Ecogenomics and potential biogeochemical impacts of globally abundant ocean viruses. *Nature*, 537(7622):689–693.

Roux, S., Emerson, J. B., Eloe-Fadrosh, E. A., and Sullivan, M. B. (2017). Benchmarking viromics: an in silico evaluation of metagenome-enabled estimates of viral community composition and diversity. *PeerJ*, 5:e3817.

Roux, S., Enault, F., Hurwitz, B. L., and Sullivan, M. B. (2015a). Virsorter: mining viral signal from microbial genomic data. *PeerJ*, 3:e985.

Roux, S., Hallam, S. J., Woyke, T., and Sullivan, M. B. (2015b). Viral dark matter and virus–host interactions resolved from publicly available microbial genomes. *elife*, 4:e08490.

Roux, S., Krupovic, M., Daly, R. A., Borges, A. L., Nayfach, S., Schulz, F., Sharrar, A., Matheus Carnevali, P. B., Cheng, J.-F., Ivanova, N. N., et al. (2019b). Cryptic inoviruses revealed as pervasive in bacteria and archaea across earth's biomes. *Nature Microbiology*, 4(11):1895–1906.

Roux, S., Krupovic, M., Debroas, D., Forterre, P., and Enault, F. (2013). Assessment of viral community functional potential from viral metagenomes may be hampered by contamination with cellular sequences. *Open biology*, 3(12):130160.

Roux, S., Matthijnssens, J., and Dutilh, B. E. (2021). Metagenomics in virology. *Encyclopedia of Virology*, page 133.

Ruby, J. G., Bellare, P., and DeRisi, J. L. (2013). Price: software for the targeted assembly of components of (meta) genomic sequence data. *G3: Genes, Genomes, Genetics*, 3(5):865–880.

Saeed, I., Tang, S.-L., and Halgamuge, S. K. (2012). Unsupervised discovery of microbial population structure within metagenomes using nucleotide base composition. *Nucleic acids research*, 40(5):e34–e34.

Salmela, L. and Schröder, J. (2011). Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461.

Salzberg, S. L., Phillippy, A. M., Zimin, A., Puiu, D., Magoc, T., Koren, S., Treangen, T. J., Schatz, M. C., Delcher, A. L., Roberts, M., et al. (2012). Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567.

Sanger, F. and Coulson, A. R. (1975). A rapid method for determining sequences in dna by primed synthesis with dna polymerase. *Journal of molecular biology*, 94(3):441–448.

Sanger, F., Nicklen, S., and Coulson, A. R. (1977). Dna sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467.

Sanjuán, R. and Domingo-Calap, P. (2016). Mechanisms of viral mutation. *Cellular and molecular life sciences*, 73(23):4433–4448.

Sanjuán, R. and Domingo-Calap, P. (2021). Genetic diversity and evolution of viral populations. *Encyclopedia of Virology*, page 53.

Sano, E., Carlson, S., Wegley, L., and Rohwer, F. (2004). Movement of viruses between biomes. *Applied and Environmental Microbiology*, 70(10):5842–5846.

Savage, D. C. (1977). Microbial ecology of the gastrointestinal tract. *Annual review of microbiology*, 31(1):107–133.

Sayers, E. W., Cavanaugh, M., Clark, K., Ostell, J., Pruitt, K. D., and Karsch-Mizrachi, I. (2019). Genbank. *Nucleic acids research*, 47(D1):D94–D99.

Schatz, M. C., Delcher, A. L., and Salzberg, S. L. (2010). Assembly of large genomes using second-generation sequencing. *Genome research*, 20(9):1165–1173.

Schirmer, M., D'Amore, R., Ijaz, U. Z., Hall, N., and Quince, C. (2016). Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC bioinformatics*, 17(1):1–15.

Schloss, P. D. and Handelsman, J. (2005). Metagenomics for studying unculturable microorganisms: cutting the gordian knot. *Genome biology*, 6(8):1–4.

Schmidt, B., Sinha, R., Beresford-Smith, B., and Puglisi, S. J. (2009). A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, 25(17):2279–2280.

Schoch, C. L., Seifert, K. A., Huhndorf, S., Robert, V., Spouge, J. L., Levesque, C. A., Chen, W., Consortium, F. B., List, F. B. C. A., Bolchacova, E., et al. (2012). Nuclear ribosomal internal transcribed spacer (its) region as a universal dna barcode marker for fungi. *Proceedings of the national academy of Sciences*, 109(16):6241–6246.

Schröder, J., Schröder, H., Puglisi, S. J., Sinha, R., and Schmidt, B. (2009). Shrec: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163.

Schulz, F., Alteio, L., Goudeau, D., Ryan, E. M., Yu, F. B., Malmstrom, R. R., Blanchard, J., and Woyke, T. (2018). Hidden diversity of soil giant viruses. *Nature communications*, 9(1):1–9.

Schulz, F., Roux, S., Paez-Espino, D., Jungbluth, S., Walsh, D. A., Denef, V. J., McMahon, K. D., Konstantinidis, K. T., Eloe-Fadrosh, E. A., Kyrpides, N. C., et al. (2020). Giant virus diversity and host interactions through global metagenomics. *Nature*, 578(7795):432–436.

Schulz, M. H., Weese, D., Holtgrewe, M., Dimitrova, V., Niu, S., Reinert, K., and Richard, H. (2014). Fiona: a parallel and automatic strategy for read error correction. *Bioinformatics*, 30(17):i356–i363.

Sczyrba, A., Hofmann, P., Belmann, P., Koslicki, D., Janssen, S., Dröge, J., Gregor, I., Majda, S., Fiedler, J., Dahms, E., et al. (2017). Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nature methods*, 14(11):1063–1071.

Sheetlin, S., Park, Y., Frith, M. C., and Spouge, J. L. (2016). Alp & falp: C++ libraries for pairwise local alignment e-values. *Bioinformatics*, 32(2):304–305.

Shendure, J. and Ji, H. (2008). Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145.

Shkoporov, A. N., Clooney, A. G., Sutton, T. D., Ryan, F. J., Daly, K. M., Nolan, J. A., McDonnell, S. A., Khokhlova, E. V., Draper, L. A., Forde, A., et al. (2019). The human gut virome is highly diverse, stable, and individual specific. *Cell host & microbe*, 26(4):527–541.

Shkoporov, A. N. and Hill, C. (2019). Bacteriophages of the human gut: the "known unknown" of the microbiome. *Cell host & microbe*, 25(2):195–209.

Simão, F. A., Waterhouse, R. M., Ioannidis, P., Kriventseva, E. V., and Zdobnov, E. M. (2015). Busco: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212.

Simmonds, P. (2015). Methods for virus classification and the challenge of incorporating metagenomic sequence data. *Journal of General Virology*, 96(6):1193–1206.

Simmonds, P., Adams, M. J., Benkő, M., Breitbart, M., Brister, J. R., Carstens, E. B., Davison, A. J., Delwart, E., Gorbalenya, A. E., Harrach, B., et al. (2017). Virus taxonomy in the age of metagenomics. *Nature Reviews Microbiology*, 15(3):161–168.

Simon, C. and Daniel, R. (2011). Metagenomic analyses: past and future trends. *Applied and environmental microbiology*, 77(4):1153–1161.

Simon-Loriere, E. and Holmes, E. C. (2011). Why do rna viruses recombine? *Nature Reviews Microbiology*, 9(8):617–626.

Slatko, B. E., Gardner, A. F., and Ausubel, F. M. (2018). Overview of next-generation sequencing technologies. *Current protocols in molecular biology*, 122(1):e59.

Smith, E. C., Blanc, H., Vignuzzi, M., and Denison, M. R. (2013). Coronaviruses lacking exoribonuclease activity are susceptible to lethal mutagenesis: evidence for proofreading and potential therapeutics. *PLoS pathogens*, 9(8):e1003565.

Smith, L. M., Sanders, J. Z., Kaiser, R. J., Hughes, P., Dodd, C., Connell, C. R., Heiner, C., Kent, S. B., and Hood, L. E. (1986). Fluorescence detection in automated dna sequence analysis. *Nature*, 321(6071):674–679.

Smits, S. L., Bodewes, R., Ruiz-Gonzalez, A., Baumgärtner, W., Koopmans, M. P., Osterhaus, A. D., and Schürch, A. C. (2014). Assembly of viral genomes from metagenomes. *Frontiers in microbiology*, 5:714.

Song, L., Florea, L., and Langmead, B. (2014). Lighter: fast and memory-efficient sequencing error correction without counting. *Genome biology*, 15(11):1–13.

Stark, R., Grzelak, M., and Hadfield, J. (2019). Rna sequencing: the teenage years. *Nature Reviews Genetics*, 20(11):631–656.

Steinegger, M., Mirdita, M., and Söding, J. (2019). Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold. *Nature methods*, 16(7):603–606.

Steinegger, M. and Söding, J. (2017). Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028.

Steinegger, M. and Söding, J. (2018). Clustering huge protein sequence sets in linear time. *Nature communications*, 9(1):1–8.

Stoler, N. and Nekrutenko, A. (2021). Sequencing error profiles of illumina sequencing instruments. *NAR genomics and bioinformatics*, 3(1):lqab019.

Suttle, C. A. (2005). Viruses in the sea. *Nature*, 437(7057):356–361.

Suttle, C. A. (2007). Marine viruses—major players in the global ecosystem. *Nature reviews microbiology*, 5(10):801–812.

Sutton, G. G., White, O., Adams, M. D., and Kerlavage, A. R. (1995). Tigr assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19.

Sutton, T. D., Clooney, A. G., Ryan, F. J., Ross, R. P., and Hill, C. (2019). Choice of assembly software has a critical impact on virome characterisation. *Microbiome*, 7(1):1–15.

Sutton, T. D. and Hill, C. (2019). Gut bacteriophage: current understanding and challenges. *Frontiers in endocrinology*, page 784.

Swerdlow, H. and Gesteland, R. (1990). Capillary gel electrophoresis for rapid, high resolution dna sequencing. *Nucleic acids research*, 18(6):1415–1419.

Takeda, H., Ueda, Y., Inuzuka, T., Yamashita, Y., Osaki, Y., Nasu, A., Umeda, M., Takemura, R., Seno, H., Sekine, A., et al. (2017). Evolution of multi-drug resistant hcv clones from pre-existing resistant-associated variants during direct-acting antiviral therapy determined by third-generation sequencing. *Scientific reports*, 7(1):1–13.

Tanaka, R., Hino, A., Tsai, I. J., Palomares-Rius, J. E., Yoshida, A., Ogura, Y., Hayashi, T., Maruyama, H., and Kikuchi, T. (2014). Assessment of helminth biodiversity in wild rats using 18s rdna based metagenomics. *PloS one*, 9(10):e110769.

Tars, K. (2020). ssrna phages: Life cycle, structure and applications. In *Biocommunication of Phages*, pages 261–292. Springer.

Thurber, R. V., Haynes, M., Breitbart, M., Wegley, L., and Rohwer, F. (2009). Laboratory procedures to generate viral metagenomes. *Nature protocols*, 4(4):470–483.

Tringe, S. G. and Hugenholtz, P. (2008). A renaissance for the pioneering 16s rrna gene. *Current opinion in microbiology*, 11(5):442–446.

Trubl, G., Hyman, P., Roux, S., and Abedon, S. T. (2020). Coming-of-age characterization of soil viruses: a user's guide to virus isolation, detection within metagenomes, and viromics. *Soil Systems*, 4(2):23.

Trubl, G., Jang, H. B., Roux, S., Emerson, J. B., Solonenko, N., Vik, D. R., Solden, L., Ellenbogen, J., Runyon, A. T., Bolduc, B., et al. (2018). Soil viruses are underexplored players in ecosystem carbon processing. *MSystems*, 3(5):e00076–18.

Turnbaugh, P. J., Ridaura, V. K., Faith, J. J., Rey, F. E., Knight, R., and Gordon, J. I. (2009). The effect of diet on the human gut microbiome: a metagenomic analysis in humanized gnotobiotic mice. *Science translational medicine*, 1(6):6ra14–6ra14.

Uyaguari-Diaz, M. I., Chan, M., Chaban, B. L., Croxen, M. A., Finke, J. F., Hill, J. E., Peabody, M. A., Van Rossum, T., Suttle, C. A., Brinkman, F. S., et al. (2016). A comprehensive method for amplicon-based and metagenomic characterization of viruses, bacteria, and eukaryotes in freshwater samples. *Microbiome*, 4(1):1–19.

Van der Kuyl, A. C. and Cornelissen, M. (2007). Identifying hiv-1 dual infections. *Retrovirology*, 4(1):1–12.

Van Dijk, E. L., Jaszczyszyn, Y., Naquin, D., and Thermes, C. (2018). The third revolution in sequencing technology. *Trends in Genetics*, 34(9):666–681.

Vázquez-Castellanos, J. F., García-López, R., Pérez-Brocal, V., Pignatelli, M., and Moya, A. (2014). Comparison of different assembly and annotation tools on analysis of simulated viral metagenomic communities in the gut. *BMC genomics*, 15(1):1–20.

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., et al. (2001). The sequence of the human genome. *science*, 291(5507):1304–1351.

Villarroel, J., Kleinheinz, K. A., Jurtz, V. I., Zschach, H., Lund, O., Nielsen, M., and Larsen, M. V. (2016). Hostphinder: a phage host prediction tool. *Viruses*, 8(5):116.

Vollmers, J., Wiegand, S., and Kaster, A.-K. (2017). Comparing and evaluating metagenome assembly tools from a microbiologist's perspective-not only size matters! *PloS one*, 12(1):e0169662.

Walker, P. J., Siddell, S. G., Lefkowitz, E. J., Mushegian, A. R., Adriaenssens, E. M., Alfenas-Zerbini, P., Davison, A. J., Dempsey, D. M., Dutilh, B. E., García, M. L., et al. (2021). Changes to virus taxonomy and to the international code of virus classification and nomenclature ratified by the international committee on taxonomy of viruses (2021). *Archives of virology*, 166(9):2633–2648.

Wang, E. T., Sandberg, R., Luo, S., Khrebtukova, I., Zhang, L., Mayr, C., Kingsmore, S. F., Schroth, G. P., and Burge, C. B. (2008). Alternative isoform regulation in human tissue transcriptomes. *Nature*, 456(7221):470–476.

Wang, Y., Pedersen, M. W., Alsos, I. G., De Sanctis, B., Racimo, F., Prohaska, A., Coissac, E., Owens, H. L., Merkel, M. K. F., Fernandez-Guerra, A., et al. (2021). Late quaternary dynamics of arctic biota from ancient environmental genomics. *Nature*, 600(7887):86–92.

Wang, Z., Gerstein, M., and Snyder, M. (2009). Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57–63.

Ward, D. M., Weller, R., and Bateson, M. M. (1990). 16s rrna sequences reveal numerous uncultured microorganisms in a natural community. *Nature*, 345(6270):63–65.

Warren, R. L., Sutton, G. G., Jones, S. J., and Holt, R. A. (2007). Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501.

Warwick-Dugdale, J., Solonenko, N., Moore, K., Chittick, L., Gregory, A. C., Allen, M. J., Sullivan, M. B., and Temperton, B. (2019). Long-read viral metagenomics captures abundant and microdiverse viral populations and their niche-defining genomic islands. *PeerJ*, 7:e6800.

Waterhouse, A. M., Procter, J. B., Martin, D. M., Clamp, M., and Barton, G. J. (2009). Jalview version 2—a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191.

Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., McGuire, A., He, W., Chen, Y.-J., Makhijani, V., Roth, G. T., et al. (2008). The complete genome of an individual by massively parallel dna sequencing. *nature*, 452(7189):872–876.

Wick, R. R., Judd, L. M., Gorrie, C. L., and Holt, K. E. (2017). Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS computational biology*, 13(6):e1005595.

Wilhelm, S. W. and Suttle, C. A. (1999). Viruses and nutrient cycles in the sea: viruses play critical roles in the structure and function of aquatic food webs. *Bioscience*, 49(10):781–788.

Williams, H. T. (2013). Phage-induced diversification improves host evolvability. *BMC evolutionary biology*, 13(1):1–17.

Williamson, K. E., Fuhrmann, J. J., Wommack, K. E., and Radosevich, M. (2017). Viruses in soil ecosystems: an unknown quantity within an unexplored territory. *Annual review of virology*, 4:201–219.

Woese, C. R. and Fox, G. E. (1977). Phylogenetic structure of the prokaryotic domain: the primary kingdoms. *Proceedings of the National Academy of Sciences*, 74(11):5088–5090.

Yan, Q., Wang, Y., Chen, X., Jin, H., Wang, G., Guan, K., Zhang, Y., Zhang, P., Ayaz, T., Liang, Y., et al. (2021). Characterization of the gut dna and rna viromes in a cohort of chinese residents and visiting pakistanis. *Virus Evolution*, 7(1):veab022.

Yang, Q., Gao, C., Jiang, Y., Wang, M., Zhou, X., Shao, H., Gong, Z., and McMinn, A. (2019). Metagenomic characterization of the viral community of the south scotia ridge. *Viruses*, 11(2):95.

Yang, X., Charlebois, P., Gnerre, S., Coole, M. G., Lennon, N. J., Levin, J. Z., Qu, J., Ryan, E. M., Zody, M. C., and Henn, M. R. (2012). De novo assembly of highly diverse viral populations. *BMC genomics*, 13(1):1–13.

Yang, X., Chockalingam, S. P., and Aluru, S. (2013). A survey of error-correction methods for next-generation sequencing. *Briefings in bioinformatics*, 14(1):56–66.

Yang, X., Dorman, K. S., and Aluru, S. (2010). Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533.

Yeom, H., Lee, Y., Ryu, T., Noh, J., Lee, A. C., Lee, H.-B., Kang, E., Song, S. W., and Kwon, S. (2019). Barcode-free next-generation sequencing error validation for ultra-rare variant detection. *Nature communications*, 10(1):1–8.

Youngblut, N. D., De la Cuesta-Zuluaga, J., Reischer, G. H., Dauser, S., Schuster, N., Walzer, C., Stalder, G., Farnleitner, A. H., and Ley, R. E. (2020). Large-scale metagenome assembly reveals novel animal-associated microbial genomes, biosynthetic gene clusters, and other genetic diversity. *Msystems*, 5(6):e01045–20.

Zeidner, G., Bielawski, J. P., Shmoish, M., Scanlan, D. J., Sabehi, G., and Béjà, O. (2005). Potential photosynthesis gene recombination between prochlorococcus and synechococcus via viral intermediates. *Environmental microbiology*, 7(10):1505–1513.

Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829.

Zerbino, D. R., McEwen, G. K., Margulies, E. H., and Birney, E. (2009). Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PloS one*, 4(12):e8407.

Zhang, H., Jain, C., and Aluru, S. (2020). A comprehensive evaluation of long read error correction methods. *BMC genomics*, 21(6):1–15.

Zhang, R., Mirdita, M., Levy Karin, E., Norroy, C., Galiez, C., and Söding, J. (2021). Spacepharer: Sensitive identification of phages from crispr spacers in prokaryotic hosts. *Bioinformatics*, 37(19):3364–3366.

Zhong, Y., Chen, F., Wilhelm, S. W., Poorvin, L., and Hodson, R. E. (2002). Phylogenetic diversity of marine cyanophage isolates and natural virus communities as revealed by sequences of viral capsid assembly protein gene g20. *Applied and Environmental Microbiology*, 68(4):1576–1584.

Zimmerman, A. E., Howard-Varona, C., Needham, D. M., John, S. G., Worden, A. Z., Sullivan, M. B., Waldbauer, J. R., and Coleman, M. L. (2020). Metabolic and biogeochemical consequences of viral infection in aquatic ecosystems. *Nature Reviews Microbiology*, 18(1):21–34.

A. Appendix

A.1. PenguiN user guide

PenguiN (protein guided nucleotide assembler) is a software to assemble short nucleotide reads. It is mainly tested on viral metagenomes, but should also be applicable for the assembly of other data types. Please note that PenguiN has been integrated into the free GPLv3-licenced Plass software. It is implemented in C++ and available for Linux and macOS. Source code and binaries can be downloaded from https://github.com/soedinglab/plass.

Installation and requirements

PenguiN is not yet part of the latest release of Plass, but can be compiled from source. Therefore, g++ (4.9 or higher) and CMake (3.0 or higher) are required.

```
git clone https://github.com/soedinglab/plass.git
cd plass
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. ..
```

PenguiN takes advantage of multicore systems through OpenMP, and uses the SIMD capabilities of the system in the redundancy reduction step with Linclust. It needs a CPU with at least the SSE4.1 instruction set to run.

Usage and command line parameter

The work outlined in this thesis has extended Plass by two more workflows: `nuclassemble` and `guided_nuclassemble`.

For the whole PenguiN workflow, run

```
plass guided_nuclassemble <INPUT> <OUTPUT> <TMP> [OPTIONS]
```

PenguiN can assemble both paired-end reads (FASTQ) and single reads (FASTA or FASTQ). For single reads pass a single file, for paired-end reads pass an even number of files.

The most important parameters are listed in the following. Parameter with type `TWIN` can always be specified independently for the first stage and second stage by setting `aa:<value>,nucl:<value>` or `nucl:<value>,aa:<value>`. If only a single value is passed, it is used for both stages. To see the full parameter overview, run `plass guided_nuclassemble` with `-h`

```
workflow parameter:
   --num-iterations TWIN       Number of assembly iterations performed on
                               nucleotide level and protein level
                               [aa:5,nucl:5]

output parameter:
   --min-contig-len INT        Minimum length of assembled contig to
                               output [1000]
   --compressed INT            Write compressed output [0]

kmermatcher parameter:
   --ignore-multi-kmer BOOL    Skip k-mers occurring multiple times (>=2) [1]
   -k TWIN                     k-mer length [nucl:22,aa:14]
   --kmer-per-seq INT          k-mers per sequence [60]
   --kmer-per-seq-scale TWIN   Scale k-mer per sequence based on sequence
                               length as kmer-per-seq val + scale x seqlen
                               [0.100]

alignment parameter:
   -e DOUBLE                   Extend sequences if the E-value is below
                               [1.000E-05]
   --min-seq-id TWIN           Overlap sequence identity threshold
                               [nucl:0.990,aa:0.970]
   --min-aln-len TWIN          Minimum alignment length [0]

cycle detection:
   --chop-cycle BOOL           Remove superfluous part of circular seq. [1]
   --cycle-check BOOL          Check for circular sequences [1]

redundancy reduction:
   --clust-min-seq-id FLOAT    Seq. id. threshold passed to linclust [0.970]
   --clust-min-cov FLOAT       Coverage threshold passed to linclust [0.990]

technical parameters:
   --max-seq-len INT           Maximum sequence length [200000]
   --threads INT               Number of CPU-cores used (all by default) [16]
   -v INT                      Verbosity level: 0: quiet, 1: +errors,
                               2: +warnings, 3: +info [3]
```

A.2. CoCo user guide

CoCo (consensus correction) is a software to correct or filter short sequencing reads. CoCo is implemented in C++ and freely available under the GPLv3-licence. The source code can be downloaded from https://github.com/soedinglab/CoCo.

Installation and requirements

CoCo can be compiled from source, pre-built binaries are not yet available. To compile CoCo from source git, g++ (5.0 or higher) and CMake (3.10 or higher) are required.

```
git clone https://github.com/soedinglab/CoCo.git
cd CoCo
git submodule update --init
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=. ..
make -j 4
```

Afterwards, the CoCo binary will be located in the build/ directory.

For optimal performance, CoCo requires a system supporting the SSE4.1 instruction set. However, it can also be compiled without.

Overview of workflows

CoCo provides different workflows:

- `correction`: Apply sequencing error correction.
- `filter`: Filter chimeric sequences.
- `abundance`: Estimate the abundance for each sequence.
- `profile`: Developer Tool to output spaced k-mer count profiles.
- `counts2flat`: Developer Tool to output the spaced k-mer count table.

Usage and command line parameter

To use a CoCo workflow, run `coco <workflow>`.

Here, only the usage and command line parameter for the sequencing error correction workflow are shown. To see the available parameters for the other workflows, run `coco <workflow>` without parameters or with `-h`

For the sequencing error correction workflow run

```
coco correction -1 <fasta|q> -2 <fasta|q> | --reads <fasta|q>
                [--counts <count.h5>] [options]
```

Specifying, the input by using either `-1 <fasta|q> -2 <fasta|q>` for paired-end sequences or `--reads <fasta|q>` for single-end reads is required. Passing a pre-computed count-table by specifying `--counts` is optional, but highly recommended. Using this option, CoCo transforms the contiguous k-mers provided in the h5 file into spaced k-mers and utilizes the memory-efficient lookuptable to store the counts. If the `counts` parameter is not specified, CoCo will utilize its internal hash-table to count the spaced k-mers in the input reads itself. However, this only works for small datasets.

An overview about CoCo's parameter is given in the following:

```
-1                  file with forward paired-end reads (fasta/fastq format)
-2                  file with reverse paired-end reads (fasta/fastq format)
--reads             file with unpaired/single/merged reads
                    (fasta/fastq format)
--counts            pre-computed contiguous kmer count file in hdf5 format
                    (dsk output)
--outdir            output directory [coco_out/]
--outprefix         prefix to use for resultfile(s)
--spaced-pattern    user-specified spaced k-mer pattern
                    [111101111110110111010101110110111111101111]
                    span must be <=64, 12 <= weight <=32 and symmetric
--skip              skip sequences with less than this many k-mers [10]
--threshold         percent drop threshold relative to neighborhood counts
                    [0.010]
--pseudocount       untrusted count added to the pseudocount parameter [1]
--lowerbound        lower bound for neighborhood counts to be considered [5]
--max-corr-num      maximal number of corrections performed per read,
                    changes are discarded otherwise [10]
--max-trim-len      maximal number of nucleotides trimmed from the beginning
                    or end of a read if error could not be corrected [0]
--update-lookup     update counts in lookuptable after a sequence
                    is corrected [0]
                    (slow down and creates read order dependency but might
                    help in low coverage regions)
--verbose           verbosity level, 0: quiet 1: Errors, 2: +Warnings,
                    3: +Info, 4: +Debug [3]
```

A.3. MMseqs2 database format

Due to the size of metagenomic samples, input and intermediate results of the described assembly processes can not be stored in memory. Therefore, it is necessary to write out these results to the disk during the assembly, which creates the bottleneck of many accesses to the file system. To tackle this, PenGuiN (and Plass) use the more efficient MMseqs2 [Steinegger and Söding, 2017] database format. The details of the format are described in the MMseqs2 User Guide (https://mmseqs.com/latest/userguide.pdf). In summary, a MMseqs2 database consists of three files (1) a data file, (2) an index file, and (3) a dbtype file. The data file contains the content of the database as \0 separated records, the index file gives for each record the corresponding byte position and size in the data file, and the dbtype file contains a number in binary format describing the type of the database e.g. amino acid, nucleotide, profile, ...

In case of a sequencing file (FASTA/FASTQ) the content is transformed into two databases, the sequence data and the header, i.e. 6 files. Each line in the sequence data file is a separate sequence followed by a null byte \0. The index file contains three tab separated columns (1) numeric sequence identifier, (2) byte offset, (3) size of the record (corresponding to the real sequencing length plus newline and \0 character). The header data file contains either the “>” or “@” records, and has its own corresponding index file. The relationship of sequence and header records are maintained by using the same identifier as key.

Besides the use of the MMseqs2 database format for sequences, also k-mer matches and alignment results are stored in the MMseqs2 database format. E.g. for each sequence id the record with the same id in the alignment database gives all alignments which were computed for this sequence.

At the same time, this format enables to store the relation between the protein sequence and the corresponding nucleotide sequences in a simple way in PenguiN, which was necessary for the simultaneous assembly of both. Both have the same numerical identifier in their respective database files. Therefore, for each protein sequence the corresponding nucleotide sequence can be queried directly. Further, with the same identifier the alignments that corresponds to the possible extension can be accessed from the protein alignment database and nucleotide alignment database.

A.4. Parameter choice for the detection of circular sequences during the assembly

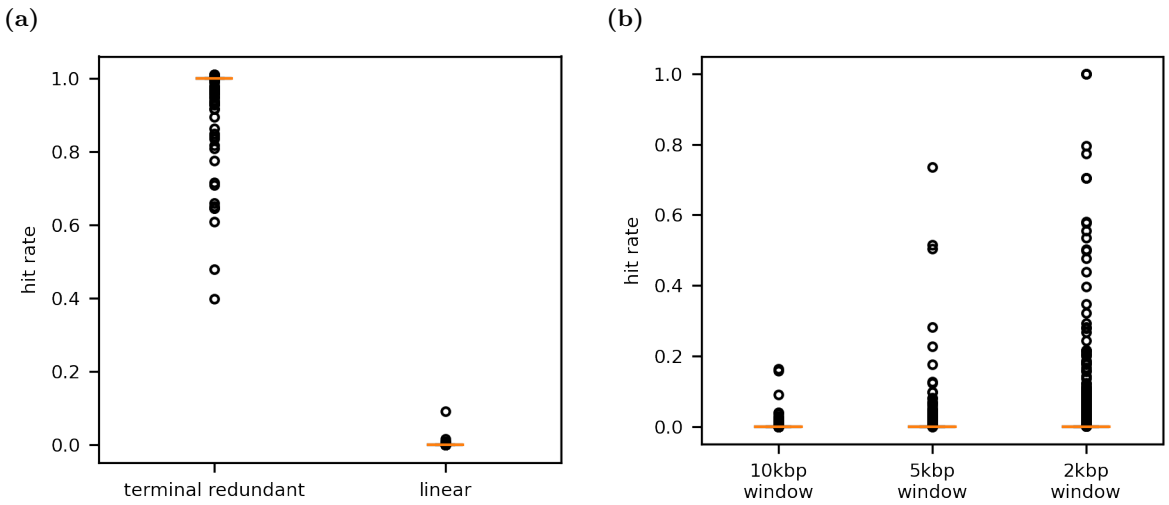


Figure A.1.: Distribution of the hit rate for terminal redundant sequences and linear sequences during PenguiN's cycle decection. Within the PenguiN software, circular sequences are identified through the number of k-mer matches between the sequences ends relative to the length (hit rate). If the hit rate is greater than a threshold, the sequence is excluded from the subsequent iterations of the assembly process to avoid overextension and only added later to the final contigs. To get a default threshold value, I analyzed the hit rate distribution for 5994 complete viruses, obtained from the RefSeq [O'Leary et al., 2016] database. By aligning the ends of the sequences I classified the viruses as terminal redundant viruses, the once that would start to repeat itself during the assembly process, and linear otherwise. (a) shows the distribution of the hit rate for the complete genomes as boxplot, and (b) the distribution of the hit rate when splitting the linear sequences in fragments to model the assembly process (for 3 different fragment sizes). The boxes (orange) contain 75% of the values and show that both classes significantly differ, there are only some outliers. I simply choose the hit rate threshold as the midpoint of the most extreme outliers of the terminal redundant and linear class from the complete genomes (0.24)

.

A.5. Resource data for the metatranscriptomic dataset

Table A.1.: Accession numbers and sample information of the metatranscriptomic dataset. The table is adapted from [Callanan et al., 2020].

SRA Accession	Location	Sample	paired reads
SRR6960803	Japan	activated sludge	20 191 321
SRR6960801	Japan	activated sludge	20 197 408
SRR6960799	Japan	activated sludge	22 422 377
SRR7976324	Japan	activated sludge	17 780 687
SRR7976357	Japan	activated sludge	23 459 568
SRR7976327	Japan	activated sludge	20 767 250
SRR7976323	Japan	activated sludge	19 691 006
SRR7976326	Japan	activated sludge	12 678 684
SRR7976325	Japan	activated sludge	17 670 226
SRR6960800	Japan	activated sludge	15 926 937
SRR6960802	Japan	activated sludge	20 208 979
SRR7976356	Japan	activated sludge	19 834 461
SRR7473382	Austria	activated sludge	50 635 570
SRR5208573	Austria	activated sludge	14 216 734
SRR6050738	Austria	activated sludge	18 337 219
SRR5208570	Austria	activated sludge	28 150 679
SRR5208572	Austria	activated sludge	29 553 670
SRR5208574	Austria	activated sludge	11 776 827
SRR5208575	Austria	activated sludge	17 963 645
SRR5467137	Austria	activated sludge	32 099 619
SRR5467139	Austria	activated sludge	24 263 621
SRR5467140	Austria	activated sludge	27 444 364
SRR6050482	Austria	activated sludge	15 846 142
SRR6050483	Austria	activated sludge	28 430 957
SRR6050528	Austria	activated sludge	21 300 721
SRR6050643	Austria	activated sludge	4 803 184
SRR6050698	Austria	activated sludge	17 703 117
SRR6050716	Austria	activated sludge	24 078 057
SRR5466399	Illinois	activated sludge	15 040 886
SRR6254352	Illinois	activated sludge	19 029 630
SRR6960507	Illinois	activated sludge	21 165 227
SRR5466338	Illinois	activated sludge	17 022 360
SRR5466364	Illinois	activated sludge	21 955 895
SRR5466365	Illinois	activated sludge	20 573 268
SRR5466366	Illinois	activated sludge	17 463 750

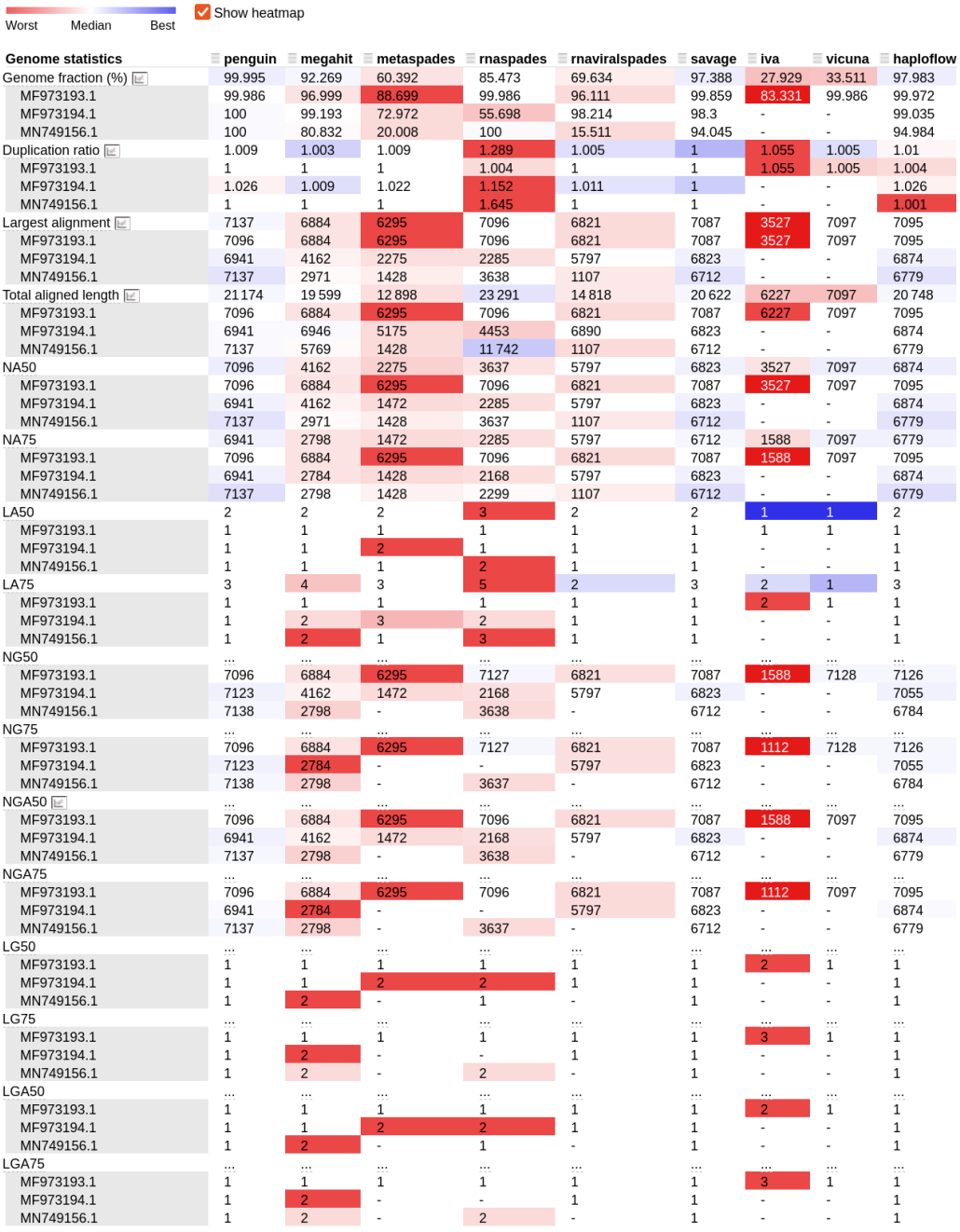
Table A.1 continued.

SRA Accession	Location	Sample	paired reads
SRR5466369	Illinois	activated sludge	16 356 361
SRR5466337	Illinois	activated sludge	19 932 210
SRR5466723	Illinois	activated sludge	15 660 162
SRR5466725	Illinois	activated sludge	16 731 551
SRR5466727	Illinois	activated sludge	13 890 630
SRR5466728	Illinois	activated sludge	15 838 965
SRR5466729	Illinois	activated sludge	13 872 521
SRR5467090	Illinois	activated sludge	16 287 026
SRR5467091	Illinois	activated sludge	15 688 657
SRR6049586	Illinois	activated sludge	16 487 854
SRR6253161	Illinois	activated sludge	19 588 661
SRR6253256	Illinois	activated sludge	16 671 597
SRR6253579	Illinois	activated sludge	19 008 214
SRR6254351	Illinois	activated sludge	18 281 569
SRR6254353	Illinois	activated sludge	16 089 478
SRR6254893	Illinois	activated sludge	17 144 853
SRR6254984	Illinois	activated sludge	17 408 755
SRR6255402	Illinois	activated sludge	20 295 615
SRR6255513	Illinois	activated sludge	16 354 936
SRR6255520	Illinois	activated sludge	14 928 927
SRR6255521	Illinois	activated sludge	18 186 323
SRR6255522	Illinois	activated sludge	14 712 936
SRR6255733	Illinois	activated sludge	15 439 553
SRR6255746	Illinois	activated sludge	18 874 520
SRR6960509	Illinois	activated sludge	21 489 598
SRR6960540	Illinois	activated sludge	665 184
SRR6960549	Illinois	activated sludge	3 322 729
SRR6960550	Illinois	activated sludge	1 801 990
SRR6960551	Illinois	activated sludge	25 867 062
SRR6960797	Illinois	activated sludge	13 437 612
SRR7976295	Illinois	activated sludge	2 568 117
SRR7976299	Illinois	activated sludge	4 197 344
SRR7976300	Illinois	activated sludge	16 710 492
SRR7976301	Illinois	activated sludge	19 019 935
SRR7976310	Illinois	activated sludge	18 395 372
SRR5834492	Lake Mendota	aquatic	20 694 213
SRR6823494	Lake Mendota	aquatic	14 858 012
SRR6435858	Lake Mendota	aquatic	25 508 743

Table A.1 continued.

SRA Accession	Location	Sample	paired reads
SRR6435866	Lake Mendota	aquatic	27 804 741
SRR7687319	Lake Mississippi	aquatic	20 624 327
SRR7687334	Lake Mississippi	aquatic	26 231 537
SRR7687308	Lake Mississippi	aquatic	25 099 737
SRR7963800	Lake Mississippi	aquatic	23 489 000
SRR5995670	Singapore	aquatic	1 586 743
SRR5995695	Singapore	aquatic	1 535 590
SRR5995666	Singapore	aquatic	1 243 067
SRR5995668	Singapore	aquatic	1 428 190

A.6. Supplementary reports



Genome statistics	penguin	megahit	metaspades	rnaspades	rnaviralspades	savage	iva	vicuna	haploflow
Genome fraction (%)	99.995	92.269	60.392	85.473	69.634	97.388	27.929	33.511	97.983
MF973193.1	99.986	96.999	88.699	99.986	96.111	99.859	83.331	99.986	99.972
MF973194.1	100	99.193	72.972	55.698	98.214	98.3	-	-	99.035
MN749156.1	100	80.832	20.008	100	15.511	94.045	-	-	94.984
Duplication ratio	1.009	1.003	1.009	1.289	1.005	1	1.055	1.005	1.01
MF973193.1	1	1	1	1.004	1	1	1.055	1.005	1.004
MF973194.1	1.026	1.009	1.022	1.152	1.011	1	-	-	1.026
MN749156.1	1	1	1	1.645	1	1	-	-	1.001
Largest alignment	7137	6884	6295	7096	6821	7087	3527	7097	7095
MF973193.1	7096	6884	6295	7096	6821	7087	3527	7097	7095
MF973194.1	6941	4162	2275	2285	5797	6823	-	-	6874
MN749156.1	7137	2971	1428	3638	1107	6712	-	-	6779
Total aligned length	21 174	19 599	12 898	23 291	14 818	20 622	6227	7097	20 748
MF973193.1	7096	6884	6295	7096	6821	7087	6227	7097	7095
MF973194.1	6941	6946	5175	4453	6890	6823	-	-	6874
MN749156.1	7137	5769	1428	11 742	1107	6712	-	-	6779
NA50	7096	4162	2275	3637	5797	6823	3527	7097	6874
MF973193.1	7096	6884	6295	7096	6821	7087	3527	7097	7095
MF973194.1	6941	4162	1472	2285	5797	6823	-	-	6874
MN749156.1	7137	2971	1428	3637	1107	6712	-	-	6779
NA75	6941	2798	1472	2285	5797	6712	1588	7097	6779
MF973193.1	7096	6884	6295	7096	6821	7087	1588	7097	7095
MF973194.1	6941	2784	1428	2168	5797	6823	-	-	6874
MN749156.1	7137	2798	1428	2299	1107	6712	-	-	6779
LA50	2	2	2	3	2	2	1	1	2
MF973193.1	1	1	1	1	1	1	1	1	1
MF973194.1	1	1	2	1	1	1	-	-	1
MN749156.1	1	1	1	2	1	1	-	-	1
LA75	3	4	3	5	2	3	2	1	3
MF973193.1	1	1	1	1	1	1	2	1	1
MF973194.1	1	2	3	2	1	1	-	-	1
MN749156.1	1	2	1	3	1	1	-	-	1
NG50	...	...	...	...	...	...	...	...	...
MF973193.1	7096	6884	6295	7127	6821	7087	1588	7128	7126
MF973194.1	7123	4162	1472	2168	5797	6823	-	-	7055
MN749156.1	7138	2798	-	3638	-	6712	-	-	6784
NG75	...	...	...	...	...	...	...	...	...
MF973193.1	7096	6884	6295	7127	6821	7087	1112	7128	7126
MF973194.1	7123	2784	-	-	5797	6823	-	-	7055
MN749156.1	7138	2798	-	3637	-	6712	-	-	6784
NGA50	...	...	...	...	...	...	...	...	...
MF973193.1	7096	6884	6295	7096	6821	7087	1588	7097	7095
MF973194.1	6941	4162	1472	2168	5797	6823	-	-	6874
MN749156.1	7137	2798	-	3638	-	6712	-	-	6779
NGA75	...	...	...	...	...	...	...	...	...
MF973193.1	7096	6884	6295	7096	6821	7087	1112	7097	7095
MF973194.1	6941	2784	-	-	5797	6823	-	-	6874
MN749156.1	7137	2798	-	3637	-	6712	-	-	6779
LG50	...	...	...	...	...	...	...	...	...
MF973193.1	1	1	1	1	1	1	2	1	1
MF973194.1	1	1	2	2	1	1	-	-	1
MN749156.1	1	2	-	1	-	1	-	-	1
LG75	...	...	...	...	...	...	...	...	...
MF973193.1	1	1	1	1	1	1	3	1	1
MF973194.1	1	2	-	-	1	1	-	-	1
MN749156.1	1	2	-	2	-	1	-	-	1
LGA50	...	...	...	...	...	...	...	...	...
MF973193.1	1	1	1	1	1	1	2	1	1
MF973194.1	1	1	2	2	1	1	-	-	1
MN749156.1	1	2	-	1	-	1	-	-	1
LGA75	...	...	...	...	...	...	...	...	...
MF973193.1	1	1	1	1	1	1	3	1	1
MF973194.1	1	2	-	-	1	1	-	-	1
MN749156.1	1	2	-	2	-	1	-	-	1

Figure A.2.: Part of the extended MetaQUAST HTML report to assess assembly quality for PenguiN and eight other assemblers on the 3-rhinovirus in silico mixture, expanded by the runs per reference. Cells differ from the median are colored. All statistics are based on contigs of size $\geq$ 1000 bp, unless otherwise noted.

Reads mapping									
# mapped	16 468	16 373	15 279	16 468	16 268	16 468	14 425	16 456	16 468
MF973193.1	16 466	16 371	15 275	16 466	16 266	16 466	14 425	16 456	16 466
MF973194.1	16 456	16 361	12 345	8903	16 256	16 273	-	-	16 456
MN749156.1	16 468	13 974	3577	16 468	2444	16 052	-	-	16 371
Mapped (%)	100	99.42	92.78	100	98.79	100	87.59	99.93	100
MF973193.1	99.99	99.41	92.76	99.99	98.77	99.99	87.59	99.93	99.99
MF973194.1	99.93	99.35	74.96	54.06	98.71	98.82	-	-	99.93
MN749156.1	100	84.86	21.72	100	14.84	97.47	-	-	99.41
# properly paired	16 468	16 168	14 800	16 468	16 076	16 354	13 946	16 456	16 224
MF973193.1	16 466	16 284	15 038	16 466	16 162	16 466	13 946	16 456	16 466
MF973194.1	16 456	16 112	11 684	8758	15 972	16 168	-	-	16 454
MN749156.1	16 468	13 624	3354	16 468	2338	15 880	-	-	16 244
Properly paired (%)	100	98.18	89.87	100	97.62	99.31	84.69	99.93	98.52
MF973193.1	99.99	98.88	91.32	99.99	98.14	99.99	84.69	99.93	99.99
MF973194.1	99.93	97.84	70.95	53.18	96.99	98.18	-	-	99.91
MN749156.1	100	82.73	20.37	100	14.2	96.43	-	-	98.64
# singletons	0	63	197	0	76	0	299	0	0
MF973193.1	0	63	195	0	76	0	299	0	0
MF973194.1	0	63	205	119	76	73	-	-	0
MN749156.1	0	286	211	0	94	122	-	-	97
Singletons (%)	0	0.38	1.2	0	0.46	0	1.82	0	0
MF973193.1	0	0.38	1.18	0	0.46	0	1.82	0	0
MF973194.1	0	0.38	1.24	0.72	0.46	0.44	-	-	0
MN749156.1	0	1.74	1.28	0	0.57	0.74	-	-	0.59
# misjoint mates	0	120	246	0	104	114	110	0	244
MF973193.1	0	0	0	0	0	0	110	0	0
MF973194.1	0	162	430	0	180	0	-	-	0
MN749156.1	0	0	0	0	0	0	-	-	0
Misjoint mates (%)	0	0.73	1.49	0	0.63	0.69	0.67	0	1.48
MF973193.1	0	0	0	0	0	0	0.67	0	0
MF973194.1	0	0.98	2.61	0	1.09	0	-	-	0
MN749156.1	0	0	0	0	0	0	-	-	0
Avg. coverage depth	115	124	174	105	163	119	336	344	117
MF973193.1	346	352	357	344	353	346	336	344	344
MF973194.1	344	348	347	293	348	353	-	-	347
MN749156.1	342	351	349	209	314	352	-	-	355
Coverage >= 1x (%)	99.15	100	100	99.8	100	100	100	100	99.52
MF973193.1	100	100	100	100	100	100	100	100	100
MF973194.1	100	100	100	100	100	100	-	-	100
MN749156.1	100	100	100	99.97	100	100	-	-	100
Coverage >= 5x (%)	98.84	99.8	100	98.2	99.76	100	100	99.69	99.33
MF973193.1	99.96	99.97	100	99.69	99.96	100	100	99.69	99.71
MF973194.1	99.72	100	100	99.87	100	100	-	-	99.73
MN749156.1	99.54	99.77	100	99.05	98.83	100	-	-	100
Coverage >= 10x (%)	98.38	99.37	100	96.64	99.35	99.7	100	99.48	99.03
MF973193.1	99.92	99.96	100	99.48	99.91	99.99	100	99.48	99.49
MF973194.1	99.51	99.96	100	99.39	99.96	99.99	-	-	99.56
MN749156.1	99.33	99.72	100	98.54	98.55	100	-	-	100

Figure A.2.: continued

(a)

Worst Median Best — Show heatmap

Genome statistics	penguin	megahit	metaspades	rnaspades	rnaviralspades	savage	iva	vicuna	haploflow
Genome fraction (%)	8.147	0.387	0.101	1.645	0.275	0.01	0.011	0.058	0.005
Duplication ratio	1.12	1.042	1.026	1.203	1.014	1	0.999	1.01	1.01
Largest alignment	9849	1730	1459	4452	3015	1170	2491	4740	1083
Total aligned length	2 066 713	88 087	23 161	445 765	62 790	2234	2491	13 379	1083
NGA50	...	...	...	...	...	...	...	...	...
LGA50	...	...	...	...	...	...	...	...	...
Misassemblies									
# misassemblies	282	69	2	112	10	0	0	3	0
Misassembled contigs length	467 948	54 280	2177	143 568	9193	0	0	5371	0
Mismatches									
# mismatches per 100 kbp	2249.32	2745.22	1552.66	2676.5	1135.84	0	4410.59	3797.47	1304.75
# indels per 100 kbp	69.56	72.33	73.32	70.19	15.91	0	40.1	164.78	0
# N's per 100 kbp	5.41	0	0	0	0	0	0	0	0
Statistics without reference									
# contigs	1139	74	20	309	49	2	1	5	1
Largest contig	13 947	2361	1459	4452	3016	1170	2491	4807	1084
Total length	2 087 455	92 163	23 795	453 490	63 753	2234	2491	13 479	1084
Total length (>= 1000 bp)	2 087 455	92 163	23 795	453 490	63 753	2234	2491	13 479	1084
Total length (>= 10000 bp)	34 566	0	0	0	0	0	0	0	0
Total length (>= 50000 bp)	0	0	0	0	0	0	0	0	0

(b)

Worst Median Best — Show heatmap

Genome statistics	penguin	megahit	metaspades	metaviralspades	rnaspades	rnaviralspades	iva	vicuna
Genome fraction (%)	66.46	14.903	4.18	0.04	29.669	10.324	0.127	0.176
Duplication ratio	1.316	1.036	1.016	1.014	1.467	1.004	1.017	1.014
Largest alignment	10 147	8190	4727	9167	7633	7594	4235	6512
Total aligned length	19 927 710	3 485 613	958 950	9167	9 794 707	2 366 627	29 055	40 213
NGA50	...	...	...	...	...	...	...	...
LGA50	...	...	...	...	...	...	...	...
Misassemblies								
# misassemblies	772	674	126	0	2362	68	10	12
Misassembled contigs length	3 090 786	943 755	114 014	0	3 509 299	91 487	21 258	22 225
Mismatches								
# mismatches per 100 kbp	955.65	1313.55	1098.23	0	3047.09	296.24	3498.02	2291.2
# indels per 100 kbp	42.16	59.54	71.56	0	104.6	14.53	72.3	64.61
# N's per 100 kbp	35.9	0	0	0	0.2	0	0	0
Statistics without reference								
# contigs	6205	2289	670	1	5969	1574	11	22
Largest contig	26 785	8191	4770	9167	7633	7594	6491	7765
Total length	20 006 079	3 533 275	971 940	9167	9 978 913	2 370 527	31 319	40 812
Total length (>= 1000 bp)	20 006 079	3 533 275	971 940	9167	9 978 913	2 370 527	31 319	40 812
Total length (>= 10000 bp)	635 265	0	0	0	0	0	0	0
Total length (>= 50000 bp)	0	0	0	0	0	0	0	0

Figure A.3.: MetaQUAST HTML report for PenguiN and nine other assemblers on the three subsets (a) 1-fold, (b) 10-fold, (c) 100-fold of the 2550-HIV1 in silico mixture. In (a) metaviralSPAdes is excluded as it did not produce any result. For the latter two, Haploflow and SAVAGE did not produce results within 10 days and are therefore excluded. Cells differing from the median are colored. All statistics are based on contigs of size ≥ 1000 bp, unless otherwise noted. Metrics that depends on the reference lengths (e.g. NGA60, LGA50) are not calculated for the combined reference.

(c)

Worst Median Best ☑ Show heatmap

Genome statistics	penguin	megahit	metaspades	metaviralspades	rnaspades	rnaviralspades	iva	vicuna
Genome fraction (%)	71.941	29.365	15.748	2.044	35.058	14.569	0.533	6.505
Duplication ratio	1.281	1.027	1.013	1.099	1.573	1.005	1.058	1.01
Largest alignment	10 054	8334	7932	9279	8697	7650	8642	4962
Total aligned length	21 012 261	6 864 428	3 621 949	513 637	12 368 580	3 340 769	123 638	1 494 105
NGA50	...	...	...	...	...	...	...	...
LGA50	...	...	...	...	...	...	...	...
Misassemblies								
# misassemblies	612	318	234	2	3080	72	83	146
Misassembled contigs length	2 919 853	716 093	245 260	6516	4 967 028	111 484	129 459	205 115
Mismatches								
# mismatches per 100 kbp	625.85	479.47	1145.81	31.66	3305.27	349.58	3070.89	671.69
# indels per 100 kbp	30.2	23.81	79.36	3.64	113.88	19.21	77.06	57.68
# N's per 100 kbp	87.54	0	0	0	6.56	0	0	5.52
Statistics without reference								
# contigs	5066	3365	2254	149	7097	2082	43	983
Largest contig	21 594	8334	7932	9279	9179	7650	19 624	5421
Total length	21 076 302	6 897 024	3 656 910	513 966	12 654 174	3 347 222	151 664	1 504 140
Total length (>= 1000 bp)	21 076 302	6 897 024	3 656 910	513 966	12 654 174	3 347 222	151 664	1 504 140
Total length (>= 10000 bp)	741 342	0	0	0	0	0	40 416	0
Total length (>= 50000 bp)	0	0	0	0	0	0	0	0

Figure A.3.: continued

Genome statistics	penguin
Genome fraction (%)	99.841
KC821629.1	99.828
KF302034.1	99.981
KF302036.1	100
KF302037.1	98.925
NC_021790.1	100
NC_021796.1	100
Duplication ratio	1.367
Largest alignment	129 415
Total aligned length	515 534
NGA50	...
LGA50	...
Reads mapping	
Mapped (%)	99.84
Properly paired (%)	99.49
Singletons (%)	0
Misjoint mates (%)	0.1
Avg. coverage depth	99
Coverage >= 1x (%)	98.42
Misassemblies	
# misassemblies	4
Misassembled contigs length	35 380
Mismatches	
# mismatches per 100 kbp	172.31
# indels per 100 kbp	10.85
# N's per 100 kbp	0
Statistics without reference	
# contigs	42
Largest contig	129 415
Total length	612 673
Total length (>= 1000 bp)	612 673
Total length (>= 10000 bp)	544 128
Total length (>= 50000 bp)	255 950

Figure A.4.: MetaQUAST HTML report for PenguiN on the mock community. All statistics are based on contigs of size ≥ 1000 bp, unless otherwise noted. Metrics that depends on the reference lengths (e.g. NGA50, LGA50) are not calculated for the combined reference. The genome fraction panel is expanded.

Table A.2.: Partial ssRNA phage genomes: contigs encoding for **at least two** phage proteins (RdRp, CP, MP), identified in the assemblies of the 82 metatranscriptomic samples. The results are reported per sample. An 'x' indicates that the assembler could not process this sample within 10 days.

Sample	PenguiN	Megahit	metaSPAdes	rnaSPAdes	rnaviralSPAdes	VICUNA	Haploflow
SRR6960803	108	99	83	68	64	x	16
SRR6960801	11	14	15	11	13	x	2
SRR6960799	229	246	243	197	171	x	70
SRR7976324	78	18	18	18	15	12	4
SRR7976357	102	94	94	78	71	x	21
SRR7976327	13	9	12	8	7	x	1
SRR7976323	47	26	25	22	20	17	10
SRR7976326	111	108	106	75	70	22	21
SRR7976325	206	193	186	176	153	45	34
SRR6960800	5	6	7	5	5	3	1
SRR6960802	62	21	22	23	19	14	10
SRR7976356	125	84	67	68	63	27	7
SRR7473382	27	31	x	32	31	x	12
SRR5208573	20	23	x	16	16	x	x
SRR6050738	25	30	30	26	26	17	19
SRR5208570	19	22	x	20	19	12	10
SRR5208572	22	21	x	20	19	x	x
SRR5208574	19	19	20	20	17	14	2
SRR5208575	27	26	26	23	23	18	15
SRR5467137	10	10	x	9	9	x	6
SRR5467139	27	28	28	26	26	22	15
SRR5467140	22	25	x	21	20	x	x
SRR6050482	9	12	13	11	12	4	4
SRR6050483	14	16	x	13	13	x	2
SRR6050528	15	16	18	15	14	8	11
SRR6050643	9	13	12	7	6	4	6
SRR6050698	24	23	23	22	22	14	15
SRR6050716	23	20	18	18	16	x	x
SRR5466399	44	30	34	30	27	9	6
SRR6254352	63	51	52	47	44	17	11
SRR6960507	119	85	83	75	66	x	11
SRR5466338	156	67	67	65	63	x	13
SRR5466364	89	73	77	72	67	x	23
SRR5466365	135	59	61	60	52	x	4
SRR5466366	99	62	60	60	54	x	10
SRR5466369	111	63	64	60	54	x	7
SRR5466337	136	67	63	64	57	x	9
SRR5466723	153	87	89	81	77	40	15
SRR5466725	175	131	141	123	119	x	73
SRR5466727	190	133	131	131	119	x	60
SRR5466728	130	74	68	70	62	37	16
SRR5466729	239	117	114	109	98	x	34
SRR5467090	185	89	85	86	75	39	17
SRR5467091	123	87	86	87	81	39	29
SRR6049586	75	62	57	59	55	x	13
SRR6253161	90	69	67	70	62	x	24
SRR6253256	55	42	42	35	31	17	12
SRR6253579	69	52	48	51	44	17	10
SRR6254351	68	47	47	44	39	19	12
SRR6254353	100	56	56	57	52	28	13
SRR6254893	64	47	44	43	44	17	16
SRR6254984	74	50	45	51	46	19	12
SRR6255402	67	46	45	48	42	20	9
SRR6255513	71	39	36	33	29	14	2
SRR6255520	60	44	40	39	34	15	8
SRR6255521	55	39	35	37	33	19	12
SRR6255522	46	40	38	38	33	12	10
SRR6255733	185	96	96	96	89	42	25
SRR6255746	99	85	88	83	78	38	46
SRR6960509	95	82	79	73	67	x	32
SRR6960540	24	14	14	15	13	7	5
SRR6960549	91	82	85	75	68	30	30
SRR6960550	27	25	24	21	20	13	12
SRR6960551	135	101	98	95	84	x	49
SRR6960797	86	59	63	58	52	22	37
SRR7976295	22	14	16	9	13	3	2
SRR7976299	115	72	73	65	58	24	17
SRR7976300	75	71	64	60	51	x	13
SRR7976301	88	73	81	62	59	23	19
SRR7976310	146	67	65	61	59	30	24
SRR5834492	2	2	2	2	2	1	0
SRR6823494	2	1	1	1	1	1	0
SRR6435858	1	1	1	1	1	x	1
SRR6435866	2	3	3	3	3	x	1
SRR7687319	3	6	8	3	1	x	1
SRR7687334	163	3	2	76	2	x	2
SRR7687308	7	9	10	6	4	x	x
SRR7963800	44	2	2	8	2	x	2
SRR5995670	29	40	38	27	39	2	3
SRR5995695	0	3	2	0	3	0	0
SRR5995666	1	1	1	1	2	1	1
SRR5995668	0	3	3	3	4	0	0

Table A.3.: Near-complete ssRNA phage genomes: contigs encoding for all **three** phage proteins (RdRp, CP, MP), identified in the assemblies of the 82 metatranscriptomic samples. The results are reported per sample. An 'x' indicates that the assembler could not process this sample within 10 days.

Sample	PenguiN	Megahit	metaSPAdes	rnaSPAdes	rnaviralSPAdes	VICUNA	Haploflow
SRR6960803	54	47	43	43	34	x	5
SRR6960801	6	7	8	6	6	x	2
SRR6960799	113	126	125	109	91	x	30
SRR7976324	34	13	13	15	14	9	2
SRR7976357	55	41	44	47	37	x	7
SRR7976327	6	4	5	4	4	x	0
SRR7976323	27	17	20	22	19	12	6
SRR7976326	56	51	53	47	36	9	7
SRR7976325	102	104	110	101	85	29	17
SRR6960800	4	4	5	4	4	1	1
SRR6960802	30	16	17	16	15	11	7
SRR7976356	65	40	41	40	33	17	2
SRR7473382	21	27	x	25	23	x	10
SRR5208573	12	13	x	12	13	x	x
SRR6050738	19	21	23	18	18	14	13
SRR5208570	13	18	x	16	15	8	9
SRR5208572	14	16	x	15	13	x	x
SRR5208574	11	12	12	11	10	7	2
SRR5208575	18	22	23	22	21	13	9
SRR5467137	6	8	x	8	8	x	5
SRR5467139	22	21	22	21	21	14	9
SRR5467140	13	16	x	14	14	x	x
SRR6050482	7	8	8	7	6	0	2
SRR6050483	10	11	x	9	8	x	1
SRR6050528	10	12	13	12	12	4	8
SRR6050643	6	6	5	6	5	3	4
SRR6050698	16	17	19	17	17	10	9
SRR6050716	15	15	14	12	10	x	x
SRR5466399	25	25	22	17	15	4	3
SRR6254352	40	38	36	34	29	11	8
SRR6960507	56	44	42	40	38	x	5
SRR5466338	85	49	47	40	37	x	5
SRR5466364	61	50	46	46	41	x	10
SRR5466365	78	45	42	40	34	x	2
SRR5466366	57	42	37	40	33	x	6
SRR5466369	58	43	41	41	35	x	3
SRR5466337	67	47	44	38	35	x	4
SRR5466723	78	62	57	56	49	22	7
SRR5466725	102	90	93	84	77	x	48
SRR5466727	100	81	83	82	75	x	37
SRR5466728	70	50	47	44	38	19	5
SRR5466729	112	73	72	74	64	x	17
SRR5467090	92	52	49	51	51	23	9
SRR5467091	76	56	59	53	44	29	15
SRR6049586	42	44	41	42	34	x	6
SRR6253161	54	50	47	47	40	x	11
SRR6253256	35	28	26	24	23	8	3
SRR6253579	43	37	32	35	29	10	7
SRR6254351	39	31	27	29	25	11	6
SRR6254353	60	41	38	34	30	15	6
SRR6254893	38	33	33	30	24	10	9
SRR6254984	43	32	30	27	23	11	5
SRR6255402	39	34	32	31	27	9	3
SRR6255513	39	28	25	18	17	6	0
SRR6255520	37	30	27	22	15	12	1
SRR6255521	33	26	24	22	21	6	7
SRR6255522	29	28	24	25	22	7	3
SRR6255733	84	66	64	63	55	25	11
SRR6255746	57	62	58	53	50	23	28
SRR6960509	58	51	51	48	41	x	17
SRR6960540	13	11	11	10	9	6	1
SRR6960549	46	50	50	46	40	21	15
SRR6960550	16	17	17	16	16	8	8
SRR6960551	73	68	67	63	55	x	28
SRR6960797	43	37	36	37	32	15	21
SRR7976295	8	4	5	4	3	3	0
SRR7976299	57	44	42	37	33	17	7
SRR7976300	41	33	35	33	25	x	6
SRR7976301	43	38	39	39	34	21	12
SRR7976310	60	42	39	36	33	25	18
SRR5834492	0	0	0	0	0	0	0
SRR6823494	0	0	0	0	0	0	0
SRR6435858	0	0	0	0	0	x	0
SRR6435866	0	1	1	0	0	x	0
SRR7687319	2	2	2	0	0	x	0
SRR7687334	31	1	2	71	2	x	2
SRR7687308	1	3	3	2	2	x	x
SRR7963800	19	2	2	8	2	x	2
SRR5995670	0	12	12	11	14	1	0
SRR5995695	0	1	1	0	1	0	0
SRR5995666	0	1	1	1	1	0	0
SRR5995668	0	2	2	2	2	0	0

Table A.4.: Complete ssRNA phage genomes: contigs encoding for all **three** phage proteins (RdRp, CP, MP) **without their premature termination by the edge of a contig**, identified in the assemblies of the 82 metatranscriptomic samples. The results are reported per sample. An 'x' indicates that the assembler could not process this sample within 10 days.

Sample	PenguiN	Megahit	metaSPAdes	rnaSPAdes	rnaviralSPAdes	VICUNA	Haploflow
SRR6960803	31	27	28	26	24	x	3
SRR6960801	3	3	4	1	2	x	1
SRR6960799	53	54	62	48	40	x	12
SRR7976324	13	8	10	7	9	5	0
SRR7976357	30	30	29	28	28	x	5
SRR7976327	3	2	2	2	2	x	0
SRR7976323	17	11	14	13	14	9	4
SRR7976326	26	19	24	21	18	3	2
SRR7976325	48	47	48	37	38	13	3
SRR6960800	2	2	1	1	1	0	0
SRR6960802	17	10	12	11	11	6	3
SRR7976356	40	24	23	25	21	12	1
SRR7473382	15	17	x	15	14	x	5
SRR5208573	7	7	x	7	7	x	x
SRR6050738	14	17	17	13	13	9	7
SRR5208570	9	11	x	9	9	5	4
SRR5208572	9	7	x	7	6	x	x
SRR5208574	8	11	10	9	9	7	2
SRR5208575	13	19	17	16	16	8	3
SRR5467137	4	4	x	5	3	x	2
SRR5467139	18	17	20	18	18	9	4
SRR5467140	11	13	x	9	12	x	x
SRR6050482	4	4	4	4	4	0	0
SRR6050483	4	6	x	5	4	x	1
SRR6050528	6	7	9	7	5	2	3
SRR6050643	2	5	4	2	1	1	0
SRR6050698	10	12	13	11	9	6	4
SRR6050716	8	12	12	7	5	x	x
SRR5466399	14	9	8	8	8	3	1
SRR6254352	21	17	15	15	14	7	0
SRR6960507	26	21	20	20	16	x	2
SRR5466338	46	26	24	23	23	x	2
SRR5466364	35	30	26	23	22	x	3
SRR5466365	50	29	21	24	20	x	0
SRR5466366	35	25	20	21	18	x	2
SRR5466369	33	24	18	19	16	x	2
SRR5466337	40	24	20	20	19	x	1
SRR5466723	44	25	25	27	25	12	3
SRR5466725	64	47	56	49	46	x	15
SRR5466727	54	41	43	44	42	x	14
SRR5466728	37	28	28	22	23	9	0
SRR5466729	58	40	35	42	34	x	5
SRR5467090	48	35	29	29	30	11	0
SRR5467091	45	34	34	36	30	14	6
SRR6049586	27	24	23	22	16	x	1
SRR6253161	29	32	27	27	23	x	3
SRR6253256	20	12	12	12	11	4	0
SRR6253579	25	17	15	18	16	5	0
SRR6254351	22	14	13	15	11	8	0
SRR6254353	32	21	18	18	12	8	1
SRR6254893	23	13	13	19	14	8	2
SRR6254984	18	13	14	13	12	6	1
SRR6255402	19	19	15	16	12	6	0
SRR6255513	23	12	9	6	7	4	0
SRR6255520	17	12	10	9	9	6	0
SRR6255521	22	15	14	11	10	4	3
SRR6255522	13	10	9	8	9	5	0
SRR6255733	41	29	36	34	29	14	5
SRR6255746	32	33	30	29	28	16	13
SRR6960509	34	19	23	25	20	x	5
SRR6960540	7	5	5	5	4	4	1
SRR6960549	26	22	27	26	25	14	2
SRR6960550	9	8	8	8	8	6	2
SRR6960551	46	37	34	32	32	x	11
SRR6960797	23	20	18	19	17	9	11
SRR7976295	4	1	1	1	1	1	0
SRR7976299	35	20	21	22	20	12	1
SRR7976300	23	21	17	19	15	x	1
SRR7976301	28	26	24	23	21	15	5
SRR7976310	39	27	25	26	27	19	5
SRR5834492	0	0	0	0	0	0	0
SRR6823494	0	0	0	0	0	0	0
SRR6435858	0	0	0	0	0	x	0
SRR6435866	0	0	0	0	0	x	0
SRR7687319	0	0	0	0	0	x	0
SRR7687334	11	1	2	44	1	x	2
SRR7687308	0	1	1	1	1	x	x
SRR7963800	7	1	1	7	2	x	2
SRR5995670	0	6	6	3	6	1	0
SRR5995695	0	0	0	0	0	0	0
SRR5995666	0	0	0	0	0	0	0
SRR5995668	0	0	0	0	0	0	0