Improved sequence-read simulation for (meta)genomics

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Stephen Johnson

Permission to Use

Abstract

There are many programs available for generating simulated whole-genome shotgun sequence reads. The data generated by many of these programs follow predefined models, which limits their use to the authors' original intentions. For example, many models assume that read lengths follow a uniform or normal distribution. Other programs generate models from actual sequencing data, but are limited to reads from single-genome studies. To our knowledge, there are no programs that allow a user to generate simulated data for metagenomics applications following empirical read-length distributions and quality profiles based on empirically-derived information from actual sequencing data.

We present BEAR (Better Emulation for Artificial Reads), a program that uses a machine-learning approach to generate reads with lengths and quality values that closely match empirically-derived distributions. BEAR can emulate reads from various sequencing platforms, including Illumina, 454, and Ion Torrent. BEAR requires minimal user input, as it automatically determines appropriate parameter settings from user-supplied data. BEAR also uses a unique method for deriving run-specific error rates, and extracts useful statistics from the metagenomic data itself, such as quality-error models. Many existing simulators are specific to a particular sequencing technology; however, BEAR is not restricted in this way. Because of its flexibility, BEAR is particularly useful for emulating the behaviour of technologies like Ion Torrent, for which no dedicated sequencing simulators are currently available. BEAR is also the first metagenomic sequencing simulator program that automates the process of generating abundances, which can be an arduous task.

BEAR is useful for evaluating data processing tools in genomics. It has many advantages over existing comparable software, such as generating more realistic reads and being independent of sequencing technology, and has features particularly useful for metagenomics work.

Acknowledgements

I would first like to thank my supervisor, Dr. Anthony Kusalik, for his enthusiasm and support for my work over the course of my degree. I thank Brett Trost, Jeffrey Long, and Vanessa Pittet for being perpetual sources of computer science and molecular biology knowledge and assisting me during the development of BEAR. An additional thanks is due to Brett Trost for his writing of the sequence identity-based abundance profile generation scripts and for allowing me to incorporate them as part of BEAR. I would also like to thank Dr. Mik Bickis and Dr. Ian McQuillan for their feedback on my thesis. Lastly, I would like to thank my parents Eric and Patricia and my younger brother Michael, whose unwavering support made graduate school far less stressful than it could have been.

Contents

List of Tables

List of Figures

List of Abbreviations

BEAR	Better Emulation for Artificial Reads
bp	base pair
CAFIE	Carry-Foward and Incomplete Extension
DNA	deoxyribonucleic acid
dNTP	deoxy ribonucleotide triphosphate
ddNTP	dideoxynucleotide
ISFET	ion-sensitive field transistor
NGS	Next-generation sequencing
nt	nucleotides
PPi	pyrophosphate
RNA	ribonucleic acid
SAM	Sequence Alignment Map
SOI	sequence of interest
SFF	sequence flowgram file
SMRT	Single molecule real time
WGS	Whole-genome shotgun
ZMW	zero-mode waveguide

Chapter 1

Introduction

A common problem in metagenomic studies is that given real data (e.g., whole genome shotgun (WGS) sequences generated by next-generation sequencing (NGS) technologies), it can be difficult to know if the bioinformatics analyses are generating useful results. In order to evaluate the results, the user typically needs to supply the bioinformatics programs (e.g., genome assembly software) with WGS sequencing data for which complete results are known. As this is often not possible in the form of real sequencing data, it is instead necessary to use artificial reads generated *in silico*.

More generally, in the field of metagenomics there are few real datasets with well-characterized, consistent results. For example, recent metagenomic studies of the human microbiome have derived results that conflict from previous studies in the same environments [82]. It is difficult to determine the usefulness of obtained results when they conflict with previously published data. It may be the case that two sets of differing results are both valid. However, it also may be the case that one set of results is invalid while the other is valid, or both sets of results are invalid. Even for problems such as *de novo* genome assembly, a simpler problem than metagenomic assembly, there is still debate as to which features make a "good" assembly due to significant variability in results between programs (e.g., high variability in average contig (contiguous assembled sequence) length and N50 values between programs) [9]. While some problem areas in bioinformatics such as multiple sequence alignment have resources like BAliBase for benchmarking [76], there are very few benchmarking datasets for metagenomics [54]. Furthermore, the simulated datasets used in previous metagenomic studies contain roughly 100 genomes, whereas actual metagenomic samples may have reads from thousands of organisms [81].

It would be far more convenient and accurate to simulate *in silico* NGS reads with known properties such that outcomes for analyses of data with these properties have been established. For example, if a simulated-read dataset is generated based on completed genomes, then various assemblers can be evaluated by determining which assembler generates contigs best matching the original genomes. Such a basis for evaluation is preferable to traditional measures such as average contig length. For software pipelines, simulated data can provide insight with respect to optimal parameter settings. Unfortunately, read simulation is not as simple as selecting random subsequences from genomes. Read length, error rates, quality scores, and community abundances (for metagenomics) can have significant variation between samples. Thus, it is important to have a tool that can emulate all of these characteristics; the tool should generate artificial data that is as “messy” as real data.

Generating *in silico* NGS reads is not without difficulties. Each NGS technology has its own error rates, quality profiles, and read-length distributions (Illumina reads are generally uniform in length, reads from other technologies can vary greatly in length). Furthermore, the technologies are constantly improving in terms of generating longer, higher-quality reads. One can easily imagine developing software that mimics a given sequencing platform, and by the time the software is complete and tested, the platform has been significantly modified by its vendor. Another inconvenience of many modern sequencing simulator programs is that the user must determine appropriate settings for numerous parameters to generate data similar to real data. Exploring the parameter space can be a serious challenge, especially if documentation is sparse. Furthermore, modern sequencing simulator programs often have fixed, internal models for characteristics such as read length distributions and quality profiles. These models may not always reflect the characteristics observed in real reads. As such, a program designed for one type of sequencer (e.g., pyrosequencer) may not adequately simulate data from another (e.g. semiconductor sequencer). When sequencing simulator programs use these fixed models, they are generally limited to simulating a specific NGS technology.

To address these shortcomings, this thesis implements, describes, and evaluates a software package called BEAR (Better Emulation for Artificial Reads) [39, 38, 37]. BEAR has, as input, a multi-FASTQ file (a file containing multiple sequences in FASTQ format) of WGS reads with the desired read length distribution and quality profile, as well as a source database. For metagenomics applications an abundance profile can be provided. BEAR generates simulated sequencing reads that are representative of genomes in the source database. The resulting data have a read length distribution and quality profile similar to those of the sample multi-FASTQ file. This approach allows for the emulation of read length distribution and quality profiles from various sequencing platforms. Since the artificial reads produced have known characteristics in terms of the source organisms and their established assemblies, the data can then be used to evaluate techniques for analysis of NGS data (such as sequence assembly or community/diversity analysis in the case of WGS metagenomic data).

Background information to the concepts presented is given in Chapter 2. The research goals of the thesis are presented in Chapter 3. Descriptions of data, design, implementation, and evaluation of BEAR are given in Chapter 4. In Chapter 5, we present the results from evaluating the data generated by BEAR and two software tests for which BEAR can be used. Chapter 6 provides the analysis and interpretation of the results, conclusions, and potential avenues for future work with BEAR.

The BEAR software was written and designed by the author of this thesis with two exceptions: the sequence identity-based abundance profile generation scripts were written by Brett Trost at the University of Saskatchewan and the majority of the modified open-source DRISEE software included with BEAR was written by Keegan et al. [41]. The work comprising Chapter 5 with the exception of Sections 5.1.4 through Section 5.4 has been accepted as a peer-reviewed paper at the RECOMB-Seq 2014 conference in Pittsburgh, PA [39], the proceedings of which will be published in an upcoming special issue of BMC Bioinformatics. The work comprising Section 5.1.4 has been presented in poster form at the HiTSeq 2014 conference in Boston, MA [37].

Chapter 2
Background

This chapter covers the necessary background material for understanding the rest of this thesis. Section 2.1 provides a brief introduction to basic concepts in molecular biology. Section 2.2 introduces next-generation sequencing, and outlines the differences between various sequencing technologies. Section 2.3 contains background information on metagenomics. Sections 2.4–2.6 introduce three important features of all data resulting from sequencing experiments: sequencing errors, quality scores, and GC bias. Section 2.7 provides a brief overview of Markov chains, and how they can be used to model data. Section 2.8 introduces the concept of sequence-read simulation, with Section 2.9 concluding the chapter with an overview of previously-developed sequencing simulator programs.

2.1 Foundation of molecular biology

Molecular biology is a branch of biology focused largely on understanding biological activity at the cellular and molecular level, particularly the structures and functions of these molecules. At a high level, molecular biology focuses largely on the activities of three types of biomolecules: DNA (deoxyribonucleic acid), RNA (ribonucleic acid), and proteins [16]. These molecules are all polymers, with a given DNA or RNA molecule being comprised of a sequence of nucleotides while proteins are comprised of a string of amino acids. There are four nucleotides that compose a DNA sequence: adenine, thymine, guanine, and cytosine. These nucleotides are commonly abbreviated by the letters A, T, G, and C, respectively [80]. RNA sequences are similar, but contain the nucleotide uracil (U) instead of thymine [2]. DNA is a double-stranded polymer of nucleotides, with each strand having a sequence of nucleotides complementary to each other in the opposite direction (a "reverse-complement" sequence) [80]. The complement of Adenine is Thymine, and the complement of Guanine is Cytosine. For example, if a strand of DNA has the sequence `AATGC`, then the reverse-complement sequence would be `GCATT`.

Eukaryotic organisms contain long, coiled strands of DNA in the nucleus of their cells known as chromosomes [42]. Prokaryotic organisms often contain small, circular DNA sequences called plasmids in addition to chromosomes [75]. Certain regions of a chromosome are copied into shorter RNA sequences by the enzyme RNA polymerase during the process of transcription. The short RNA sequences themselves are often referred to as 'transcripts'. Many of these RNA sequences, called "messenger RNA" or "mRNA" are then translated into sequences of amino acids by ribosomes (a large complex of RNAs and proteins). This process is known as translation, with the new amino acid sequence eventually becoming an active protein. This flow of information, from DNA to RNA to protein, is commonly referred to as the central dogma of molecular biology [16]. However, it is important to note that not all regions of DNA eventually become translated into a protein sequence. Only certain regions of a DNA sequence "code" for a functional RNA or protein sequence, and these regions are commonly referred to as "genes" [74]. The entire DNA sequence of all the chromosomes (and plasmids) of an organism, then, is called the "genome" [44]. Similarly, the sum of all RNA transcript sequences in an organism is called the "transcriptome".

While the composition of DNA was identified in 1878 [18] and the structure identified in 1953 [80], it was not until 1976 that the first genome sequence of an organism, the viral RNA genome of bacteriophage MS2 was established [25]. In 1977, Fred Sanger completed the first DNA genome of the bacteriophage ΦX174 by developing a sequencing method known by its technical name, dideoxynucleotide chain-termination sequencing, and by its common name, Sanger sequencing [67]. The actual process of Sanger sequencing is quite simple, requiring only five types of molecules:

- a DNA sequence of interest (SOI),
- DNA polymerase, the enzyme responsible for DNA replication,
- the four standard deoxynucleotides (dNTP), used for replicating the SOI,
- modified versions of the dNTPs known as di-deoxynucleotides (ddNTP), which terminate the replication process upon incorporation,
- a DNA primer, a short strand of nucleic acid that acts as the starting point for DNA replication.

Four reactions take place, each containing just one ddNTP, but otherwise containing all of the above. The four reactions are then subjected to gel electrophoresis, where they can be sorted by weight and the nucleotide at a given position in the sequence can be identified by the electrophoresis lane it is in. A DNA fragment of up to 800 nucleotides (nt) or basepairs (bp) can be sequenced at once using this method, with approximately the first 600 bases being of high quality. An example of a short 6bp read resulting from Sanger sequencing can be seen in Figure 2.1. Given that the actual nucleotide sequence of the SOI is determined by "reading" the gel, the sequenced fragments resulting from a sequencing experiment are commonly referred to as reads.

The Sanger method is known for its accuracy, and was the most popular sequencing method for decades after its inception. It is still in use today for some applications requiring long sequence reads, although it

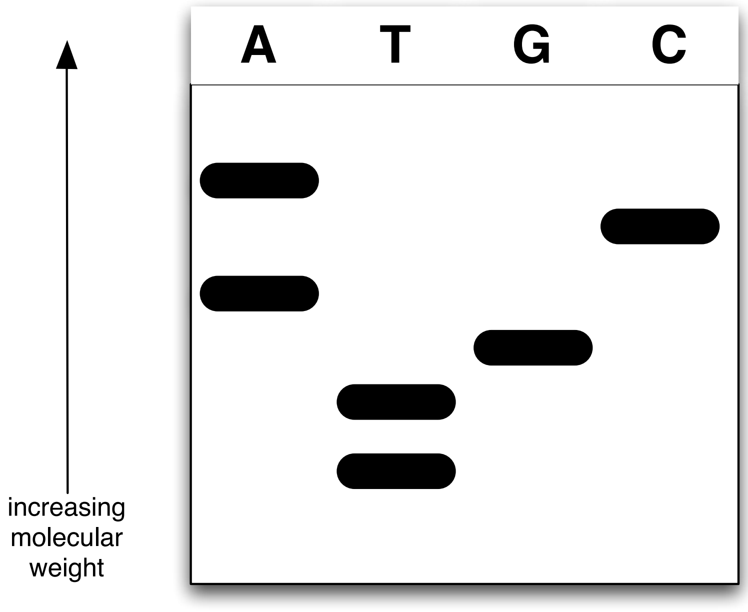


Figure 2.1: Example of gel resulting from Sanger sequencing. The sequence associated with this 6bp read would be TTGACA, a known bacterial promoter sequence.

has largely been replaced by the next-generation of sequencing technologies.

2.2 Next-generation sequencing

By the end of the 20th century, the first rough draft of the human genome was nearing completion. By this time, the parallelization of Sanger sequencing had allowed for massive amounts of data to be generated in far shorter amount of time. For comparison, in 2000 there was a total of 8 billion base pairs of genomic sequence information in the the main databases for completed sequences. By 2010, with the adoption of these less-expensive, high-throughput sequencing methods, this number had increased to 270 billion base pairs [22]. In this section, we provide an overview of the three most popular next-generation sequencing (NGS) technologies: pyrosequencing developed by Roche/454, reversible dye sequencing developed by Illumina, and semiconductor sequencing developed by Ion Torrent. We conclude the section with a brief mention of promising emerging sequencing technologies, such as single molecule real time (SMRT) sequencing by PacBio.

2.2.1 Pyrosequencing

Pyrosequencing (also called 454 sequencing) was developed in 1996 by Roche/454 to address the need for robust, high-throughput alternatives to Sanger sequencing [64]. In contrast to the chain-termination method used in Sanger sequencing, pyrosequencing adopts a "sequencing-by-incorporation" approach. That is, a given strand of DNA can be sequenced merely by synthesizing the complementary strand and measuring the intensity of some chemical byproduct during the nucleotide incorporation. Pyrosequencing requires the same materials as Sanger sequencing, excluding the ddNTPs, in addition to three enzymes (ATP sulfurylase, luciferase and apyrase) and two substrates (adenosine 5' phosphate and luciferin).

The sequencing reaction begins by incubating the primer with a single-stranded DNA sequence. The sequence is then subjected to a series of dNTP "flows", where only one type of nucleotide is added to the reaction and then "washed" away. For example, if the flow order for a given sequencing reaction was ATGC, dATP would be added to the reaction, washed away, followed by dTTP, dGTP, dCTP, and then the cycle would continue with dATP again.

When a nucleotide is added to the reaction and is incorporated into the sequence of interest during the pyrosequencing process, a cascade of reactions occur. When DNA polymerase forms a phosphodiester bond between the complementary nucleotide and a DNA sequence, a pyrophosphate (PPi) molecule is released. When ATP sulfurylase is in the presence of adenosine 5' phosphate, it converts PPi to ATP. This ATP allows luciferase to convert luciferin to oxyluciferin, a process that generates light proportional to the amount of ATP involved. The apyrase enzyme then degrades the remaining ATP and nucleotides, "washing away" the extra molecules so that the next flow reaction can begin. The light generated by the successful incorporation of a nucleotide is measured by a camera and stored in an SFF (sequence flowgram file) so that the data can later be analyzed. An example of a sequence generated by pyrosequencing is provided in Figure 2.2. Information regarding the SFF format is provided in Section 2.4.

Pyrosequencing was once a preferred technique as it is a faster, less-expensive alternative to Sanger sequencing with long read lengths relative to other NGS platforms. Pyrosequencing could produce sequences with and average read length of 700bp (ranging from 1-1200bp), generating roughly 1 million base pairs per day at a cost of $10 [50]. However, it has since fallen out of favour to even faster, less-expensive techniques like semiconductor and reversible dye sequencing. Roche, the major licensee of the pyrosequencing technology, announced in 2013 that the platform would be discontinued by 2016 [26].

2.2.2 Ion semiconductor sequencing

Ion semiconductor sequencing was developed in 2010 by Ion Torrent Systems as a fast, inexpensive alternative to pyrosequencing [66]. Much like pyrosequencing, ion semiconductor sequencing is also a "sequencing-by-incorporation" technique, but relies on the detection of hydrogen ions (H^+) upon successful nucleotide incorporation rather than pyrophosphate molecules. These ions change the pH of the reaction solution, which

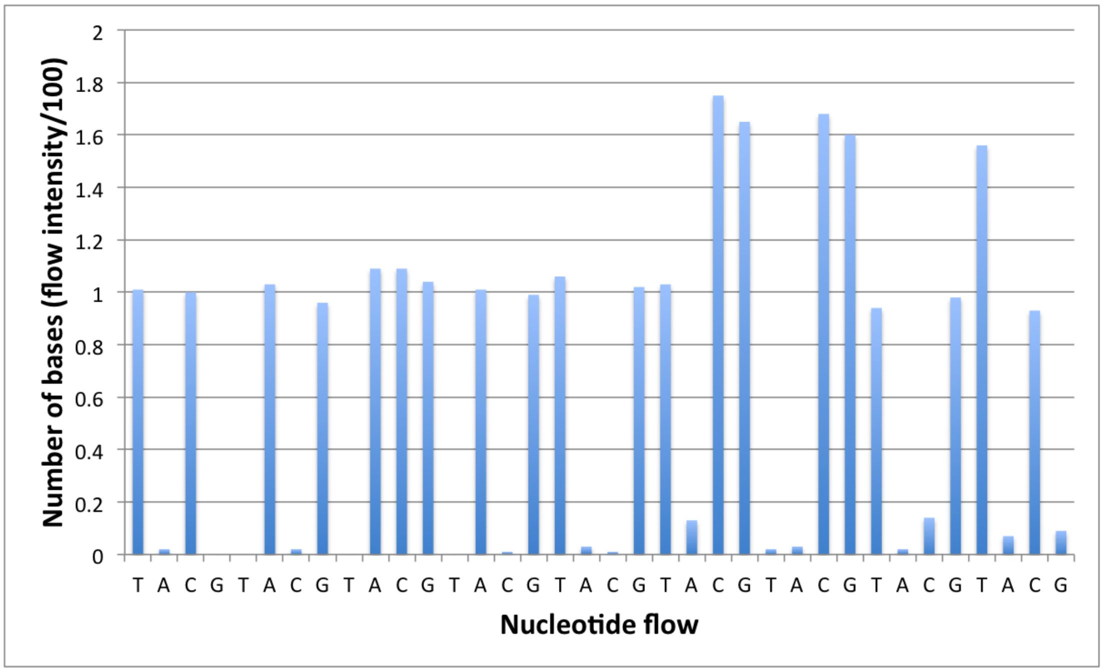


Figure 2.2: An example of a sequence generated from a pyrosequencing reaction. Peak signals correspond to incorporated nucleotides. This flowgram would correspond to the sequence TCACACGAGTGTCCGGCCGGTGTTC.

is detected by an ion-sensitive field transistor (ISFET). The current flowing through the ISFET changes with the pH of the surrounding solution. The sequencer measures the ISFET current over the course of the sequencing reaction, which is stored in an SFF file much like that used in Figure 2.2, only the y-axis would be current intensity instead of light intensity.

Ion semiconductor sequencing has grown in popularity since its inception due to its speed, low cost, and high throughput. Approximately 80 million reads can be generated in 2 hours for $1, although the sequences have shorter read lengths (up to 400bp), and a per-base accuracy of approximately 98.5%. For comparison, pyrosequencing and Illumina sequencing have per-base accuracies up to 99.9% [61].

2.2.3 Illumina/Reversible dye terminator sequencing

Reversible dye sequencing (also called Illumina sequencing, after the company who owns the technology) was developed in the mid-1990s as a way to generate high-quality, short reads (per-base accuracy above 99.9%, reads shorter than 100bp) with high throughput at low cost (up to 3 billion reads per run, at a cost of less than $0.15 per million bp) [61, 8]. Unlike the two previously described methods, Illumina sequencing uses a “sequencing-by-synthesis” approach. This approach directly determines the sequence of a DNA strand by reading the dye group attached to a flourescently labelled nucleotide, rather than measuring the chemical byproduct of nucleotide incorporation (e.g., hydrogen ions, light resulting from a separate enzyme reaction), which requires the use of expensive enzymes.

There are two primary steps to most Illumina sequencing experiments:

1. **Colony preparation/Bridge PCR**: See Figure 2.3. The sequence of interest has two different adapter sequences ligated to each end. This sequence is then hybridized to an array of bound primer sequences that are complementary to each adapter. The sequence is then synthesized, so that there is now a copy of the sequence bound to the array. The original read is then removed by the process of denaturation. The new sequence then hybridizes to another primer on the array, and then undergoes another round of synthesis and denaturation. At this point, there are now two sequences bound to the array at different primer sequences, forming a small "colony" of identical sequences. This process can be repeated for many rounds to amplify the amount of available data.

2. **Sequencing:** See Figure 2.4. After the clonal colonies have been sufficiently amplified, one of the adapter sequences is cleaved ('nicked') and denatured. The DNA is then sequenced from the point starting at this 'nicked' adapter. All four nucleotides are added at once, with each type labelled with a distinct fluorescent marker and each competing to bind to the sequence of interest. The nucleotide complementary to the current position of the sequence binds to the SOI, and the remaining nucleotides are washed away. A laser then 'reads' the fluorophores, determining the exact nucleotide sequence. The template hybridizes to the array again, and the process repeats starting from the opposite adapter. This results in a pair of reads each sequence, commonly referred to as *paired-end sequencing*.

2.2.4 Prospective technologies

Current sequencing platforms require a large amount of sequence data in order to generate reliable results due to short read lengths resulting from termination and measuring enzymatic activity [23]. Two emerging technologies, single-molecule real-time (SMRT) sequencing and nanopore sequencing, address this problem by sequencing an entire molecule of DNA.

SMRT sequencing has been in development by Pacific Biosciences since 2003, and requires only a zero-mode waveguide (ZMW), DNA polymerase, and flourescently labelled nucleotides to fully sequence a DNA molecule of up to 30,000bp. The ZMW can be thought of as a nano-chamber that is extremely sensitive to light with a DNA template-polymerase complex bound to it. The labelled nucleotides are added to the chamber and bind to the template-polymerase complex upon incorporation. Incorporation of a nucleotide produces a small pulse of light, much like pyrosequencing. However, the small chamber volume provides 1000-fold improvement in the reduction of background noise relative to other NGS methods. [23]. This

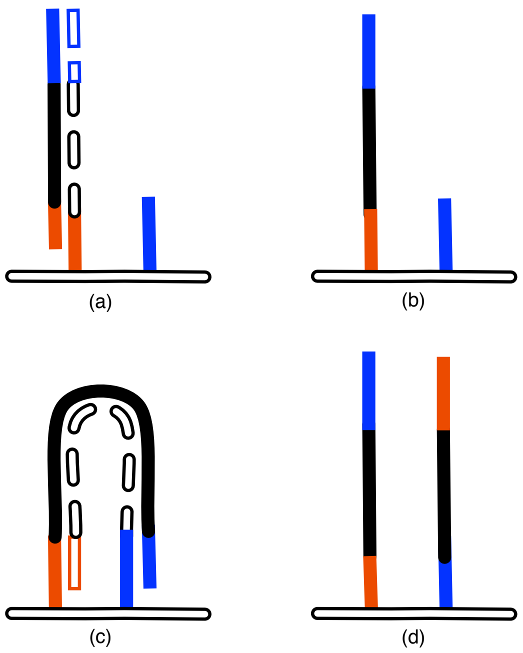


Figure 2.3: Example of Illumina single clonal preparation. Dotted lines represent new sequences. (a): SOI with adapter sequences is ligated to array of oligonucleotide primers and sequenced. (b): The sequence is denatured, unbinding the original sequence from the array. : The new sequence binds to another primer on the array and is synthesized again. (d): The array is denatured. Figure uses information from Bentley et al. [8].

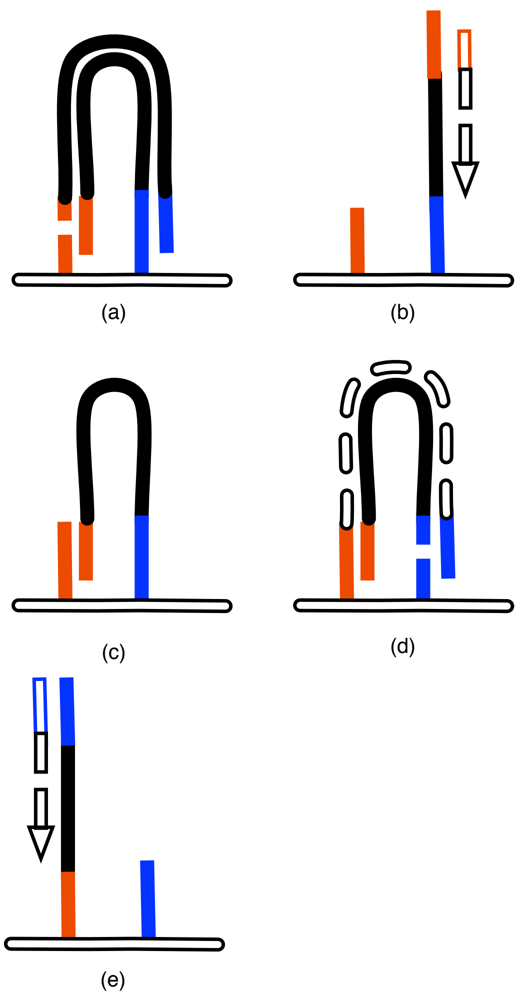


Figure 2.4: Example of Illumina paired-end sequencing following clonal preparation. (a) One adapter is cleaved (denoted by gap). (b) DNA is denatured and sequence. Dyed nucleotides compete for positions on the strand. (c) Sequence is hybridized to array again. (d) Sequence is copied, and the opposite adapter is cleaved. (e) The DNA is then sequenced from the opposite end, resulting in a pair of reads. Figure uses information from Bentley et al. [8].

method has successfully been used to generate reads with an average length of 8,500 bp, almost an order of magnitude larger than other modern technologies [29]. Unfortunately, the throughput of this technology has been known to be low (less than 100,000 reads per run) and expensive (up to $1 per million bp) relative to other current technologies [61].

While SMRT sequencing shares many features with other modern technologies (e.g., using similar enzymes), the same is currently not true for nanopore sequencing. Nanopores are small holes, typically found in transmembrane proteins. As a given DNA sequence is forced through the nanopore one base at a time, the amount of current that can pass through the pore depends on the nucleotide occupying the pore [40]. This approach is a paradigm shift in terms of how a sequencing experiment can be conducted, as there may be no need for fluorescent nucleotides or sequence amplification. However, despite being in development since 1995 and promising preliminary experiments, nanopore sequencing is not commercially available as of 2014. [40, 83].

2.3 Sequencing errors

None of the previously described sequencing technologies have been able to generate reads with 100% per-base accuracy. In this section, we provide an overview of the different types of sequencing errors one encounters with the three technologies described in Sections 2.2.1–2.2.4.

1. **Substitution errors** occur when a given nucleotide in a DNA strand is replaced with a different nucleotide at the same position. Substitution errors can either be **transitions** (A $\leftrightarrow$ G or C $\leftrightarrow$ T) or **transversions** (A $\leftrightarrow$ C, A $\leftrightarrow$ T, G $\leftrightarrow$ C, or G $\leftrightarrow$ T). Transition errors have been reported to be more common in pyrosequencing and Illumina data, and equal to transversion errors in Ion Torrent data [11, 30, 19, 10]. Additionally, substitution errors are the most common sequencing error type in Illumina data [19]. An example of each type of substitution error can be seen in Figure 2.5(a).

2. **Insertion and deletion (indel) errors.** Insertion errors occur when a nucleotide is incorporated into the sequence that is not part of the original strand, increasing the overall length of the sequence by at least 1bp. Deletion errors are the opposite of insertion errors; they occur when a nucleotide is removed from the sequenced strand, decreasing the overall length of the sequence by at least 1bp. Insertion and deletion errors are commonly referred to as *indels*. Indel-type errors are the most common sequencing errors in pyrosequencing and Ion Torrent data, but are rare in Illumina data [11, 19, 10]. Examples of indel errors can be seen in Figure 2.5(b).

3. **Homopolymer errors** are interpretation errors that occur in both pyrosequencing and Ion Torrent

data [11, 10]. Many DNA sequences have regions where the same base occurs several times in a row (e.g., `AAAA`, `GGG`, etc.), known as 'homopolymeric regions'. Thus, these regions are sequenced in a single 'flow' and result in a high intensity light or electrical signal. However, the intensity of signals is not necessarily linear. For example, in Figure 2.2, there is a peak at the last thymine flow with an intensity of 1.56. Thus, this light signal could either be interpreted as a homopolymer of length two (`TT`) or length one (`T`) depending on the cutoff values used for the signal. Since Ion Torrent and pyrosequencing depend on successful interpretation of signals to determine a given DNA sequence, these technologies have difficulties sequencing even short homopolymeric regions. In one experiment, over 97% of the measured sequencing errors in an Ion Torrent dataset were found to be homopolymer-related [10]. An example of a homopolymer sequencing error can be seen in Figure 2.5(c).

4. **Carry-Foward and Incomplete Extension (CAFIE) errors** are another type of indel error that occurs in Ion Torrent and pyrosequencing. Carry-forward errors occur when sequencing wells are not sufficiently 'flushed' and leftover nucleotides are present. Conversely, incomplete extension errors occur when the amount of nucleotide within a flow is too low. Both of these errors can cause a read to get 'out of sync', resulting in homopolymer-like errors being incorporated into the final sequence that are not related to the current nucleotide(s) in the reference sequence [5].

The effects of sequencing errors can be reduced by repeatedly sequencing a genome, achieving a higher "depth" or "coverage", referring to the average number of times each base pair in the genome is sampled. Genome coverage can be estimated by the following formula:

$$Coverage = \frac{(Avg.\, read\, length)(Number\, of\, reads)}{Length\, of\, genome} \tag{2.1}$$

Additionally, it has been noted that the error rate tends to increase towards the end of reads for modern NGS technologies, although the degree to which the rate increases can depend on a number of variables ranging from the exact sequencing preparation method used to the global GC content (see Section 2.5) of the genome of interest [10, 68, 52].

2.3.1 Quality scores

Given that NGS technologies are error-prone, there needs to be some sort of confidence measure to indicate when there has been an aberration in a sequencing experiment. Modern NGS technologies currently use "Phred quality scores", an integer value assigned to each nucleotide base call as a measure of predicted accuracy [24]. A quality score is assigned to a given nucleotide by analyzing characteristics of the shape of

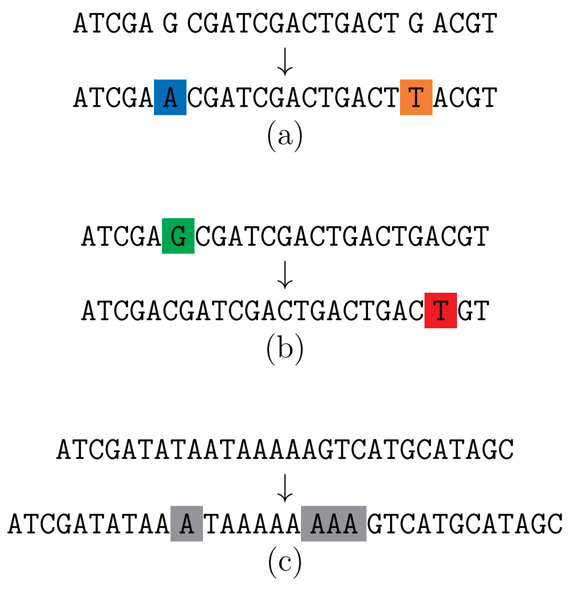

ATCGA G CGATCGACTGACT G ACGT
↓
ATCGA A CGATCGACTGACT T ACGT
(a)
ATCGA G CGATCGACTGACTGACGT
↓
ATCGACGATCGACTGACTGAC T GT
(b)
ATCGATATAATAAAAAGTCATGCATAGC
↓
ATCGATATAA A TAAAAA AAA GTCATGCATAGC
(c)

Figure 2.5: Examples of sequencing errors. (a) Substitution errors. The blue nucleotide signifies a transition error, while the orange nucleotide signifies a transversion error. (b) Indel errors. The green nucleotide has not been incorporated into the final sequence, and the red nucleotide has been inserted. (c) Homopolymer errors. Grey nucleotides are the result of incorrect signal interpretation on behalf of the sequencer and not actually incorporated into the molecule itself.

the flow signal (e.g., peak signal intensity, peak signal resolution) followed by searching through large lookup tables for scores that match these characteristics. The values in the lookup tables themselves are derived from sequences that are known *a priori*. The quality score of a given base call is logarithmically related to the probability of an incorrect base call by the following formula:

$$Q = -10 \log_{10} P \tag{2.2}$$

Conversely, the probability of a base call being correct can be determined given a quality score:

$$P = 1 - 10^{\frac{-Q}{10}} \tag{2.3}$$

For example, a quality score of 20 would indicate 99% accuracy and a score of 40 would indicate 99.99% accuracy. Thus, in a set of 100bp Illumina reads with an average quality score of 20, one would expect, on average, one error per read. Because of this, reads with average quality scores below 20 are often discarded.

However, quality scores are not perfect. Given that they are only assigned to sequences within a read, it is not possible to determine whether an insertion or deletion error has occurred [10]. Additionally, quality scores have been demonstrated to have a strong influence over the quality scores of adjacent nucleotides

```
>AB000263 |acc=AB000263|descr=Homo sapiens mRNA for prepro cortistatin like peptide
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

Figure 2.6: An example of a sequence in FASTA format.

[55, 34] and sometimes overestimate or underestimate the true underlying error rates [51].

2.4 Common file formats for sequence analysis

In this section, we provide a brief overview of three ubiquitous file formats for storing sequence data: the FASTA, FASTQ, SAM, and SFF formats.

2.4.1 FASTA

The FASTA format (short for "fast alignment') is a simple text-based format for storing DNA, RNA, or protein sequence data. When many sequences in FASTA format are concatenated in a file, this is commonly referred to as a "multi-FASTA" file. Each sequence in FASTA format has two elements: a header, and a sequence. The header is prefixed with the `>` (greater-than) symbol, and can contain known information about the sequence. FASTA headers can only be one line of characters in the file for each sequence. The sequence is simply the raw nucleotide or amino acid sequence, which can be any number of lines. An example of a sequence in FASTA format can be seen in Figure 2.6.

2.4.2 FASTQ

The FASTQ format is a more modern version of FASTA. In addition to the header and sequence information, FASTQ format provides a second line for descriptions (prefixed with the `+` (plus sign) character) and a string of quality scores for each nucleotide. These are integers converted to characters in ASCII format according to the ASCII encoding, so any quality score from 0 to 93 can be encoded and represented in FASTQ format by converting to an ASCII character with a value from 33 to 126. More information on quality scores can be found in Section 2.3.1. An example of a sequence in FASTQ format is provided in Figure 2.7.

```
@G4CQRNT01BG1K3
cagacgagtgcgtCCGGCCGGTGTTCATCCGACGCGAATTGGTGCGATTATAAGCGGCACGGTGGCACTGAACCTTACGCTATCCTGAT
+
FFFFFFFFFFDBB?10000...42229>><<<?BBDD8668??DBBBBBDF888<ADDDFFFCCDDFFFFFFFFFFFFFFFFFFDBBAA
```

Figure 2.7: An example of a sequence read in FASTQ format. The lower-case nucleotides at the beginning of the sequence is the adapter sequence.

2.4.3 SAM

The sequence alignment map (SAM) format is a commonly used format for storing information regarding the alignment of sequencing reads to genomes. It is a relatively compact format, and can be easily converted to FASTA/FASTQ format. Each sequence in SAM format consists of twelve mandatory fields in tab-delimited text format [45]:

1. The name of the read or read pair.
2. A bitwise flag that indicates various characteristics of the read (e.g., whether the read is first or second in a pair, fails quality checks, unmapped, etc.).
3. The name of the aligned reference genome/sequence.
4. The first position of the genome to which the read aligns.
5. An overall "mapping quality" value, the value of which is depending on the specific alignment tool used.
6. A string in CIGAR format (described in detail in Section 4.3.3), which indicates which positions of the read are matches (e.g., not an indel error), insertions, or deletions.
7. The name of the aligned reference genome for the other read in the pair.
8. The first position of the genome to which the other read in the pair aligns.
9. Estimated insert (insertion) size.
10. The nucleotide sequence of the read.
11. The quality string associated with the read. The format of the string is identical to FASTQ quality strings.
12. A field consisting of any number of optional fields. This often contains less important, aligner-specific information.

2.4.4 SFF

Sequence flowgram format (SFF) files are generated by the sequencing machinery itself. These files are in binary format, and thus require a proprietary program (provided with the sequencer) to view them. For each sequence, an SFF file contains the flowgram (see Figure 2.2 for a graphical interpretation of a flowgram), the predicted sequence, the quality scores, and the recommended parameters for quality and adaptor clipping.

2.5 GC bias

Genomes are frequently referred to by their GC content (the proportion of the genome made up of guanine and cytosine). It would be expected that the GC content of a given organism would be approximately 50%, but genomes do not consist of uniformly-distributed nucleotides, and the nucleotides themselves are not necessarily present in equal amounts. That is, it is not usually the case that a given genome has $\%A = \%T = \%G = \%C$. Rather, organisms have been found to have global GC contents ranging from 16% to 75% [48]. Even across the length of a given genome, there is often high variance in local GC content [35].

Sequencing these GC-rich and GC-poor genomes and genomic regions is a known issue for many sequencing technologies, protocols, and preparatory stages of sequencing experiments [13]. In sequencing experiments, there is a often a bias towards or against regions of high or low GC content high which leads to uneven sequencing coverage, necessitating extremely high sequencing depths achieve a sufficient minimum depth to avoid confounding downstream processing and analysis [1, 17]. This sequencing bias, or "GC bias" can be introduced at any stage of the sequencing experiment (colony amplification, sequencing, processing) and changes to protocols and sequencing kits can further introduce bias. Additionally, GC bias can vary between laboratories, sequencing experiments, or even lanes on the same flowcell [1]. Variables involved in the amplification stage such as the specific type of DNA polymerase used, cycle temperature, are thought to be the major contributors to GC bias, although the exact mechanisms are not yet well-understood [17].

2.6 Metagenomics

Metagenomics is commonly defined as the sequencing of genetic material obtained from environmental samples (e.g. lakes, soil, human gut) [81]. This is in contrast to traditional, singular genomic studies, which usually require organisms to be cultured in a lab so that enough genetic material can be collected to sequence the entire genome with high coverage (coverage is defined as the average number of reads that 'overlap' at a given nucleotide position; see Equation 2.3). Additionally, growing the organism in culture allows sequence data to be easily verified (i.e., when some region of DNA is sequenced, it must be known that this sequence came from the organism of interest). Unfortunately, not all organisms can be grown in culture. In fact, it has been conservatively estimated that the majority of microorganisms cannot be cultured [62]. Metagenomic studies allow for the characterization of entire microbial populations and environments (e.g., relative species abundance profiling) as a function of their genomes, a feat that would be impossible with traditional sequencing and culture methodologies.

The major challenges in metagenomics are intimately tied to current challenges in bioinformatics and next-generation sequencing. Both metagenomic and bioinformatic studies face the difficulty of organizing and analyzing huge amounts of data, a phenomenon commonly referred to as a biological "data explosion" [69]. In these two fields, there are millions of DNA sequences being generated faster than the data can be fully analyzed. Since much of bioinformatics is concerned with the analysis and collection of biological sequences, adopting a bioinformatics approach (e.g., using bioinformatics tools) is necessary for most metagenomic experiments.

Many problems in metagenomics analysis are also related to problems with the currently available sequencing technologies. For example, it is uncommon to obtain high sequencing depth of metagenomic samples from even moderately diverse environments. There are steep time and space requirements to achieving high sequencing depth in metagenomic studies due to the immense amount of genomic data in a given sample. For example, if we conservatively assume that a sample has 1000 organisms in equal abundance with an average genome size of 4.5 million base pairs and our NGS machine generates reads with a length of 100bp, it would take 225 million high-quality reads just to achieve 5.0x coverage (i.e., on average, every nucleotide in every sequence is present at least 5 times). Depending on the sequencing technology used, this could take between hours and weeks. Although NGS techniques can generate many millions of reads, a significant amount of these reads are low-quality artifacts generated by the sequencer and can be discarded [28]. Thus, the immense computational resources and storage required to obtain a sufficient volume of high coverage, high quality data may not always be cost-effective.

There are two main types of metagenomic sequencing experiments: whole-(meta)genome shotgun (WGS) sequencing, and amplicon sequencing. In WGS experiments, the raw sequence data is fragmented into smaller sequences, which are then subjected to some type of sequencing protocol. The resulting reads (which are smaller than the fragment size) can then assembled into larger, contiguous sequences (contigs). This approach is preferable when estimating abundance profiles (i.e., the relative abundances of species in a metagenomic sample), or when determining the functional characteristics of organisms in the environment, as many whole genomes are being sequenced at once [59]. Additionally, one could perform whole-(meta)transcriptome shotgun sequencing on the RNA present in the sample to determine which genes are actively being transcribed.

Amplicon sequencing in metagenomics generally involves the sequencing of a gene found to be highly conserved within individual microbial species, usually the 16S ribosomal subunit RNA gene (16S rRNA) [59]. This is in stark contrast to WGS experiments, in which potentially thousands of genes are sequenced. Primers specific to the 16S rRNA gene are added to the sample, which is then subjected to amplification via polymerase chain reaction (PCR) [7]. Amplification is necessary to increase the abundance of this gene relative to other genes, and allow for more 'sanitary' data (e.g., reduce the effects of sequencing errors). The end result of amplicon sequencing is a number of unique sequences that should, ideally, directly correspond to the number of unique species present in the sample. However, since the data has been subjected to amplification, there is no guarantee that the relative abundances of the organisms has been preserved or that the sequences are not altered.

2.7 Community profiling

Community profiling is a common task performed on shotgun metagenomic data. Given that shotgun data is often unamplified (unlike amplicon or genomic data), the abundances of organisms are preserved. This allows for the diversity of the community to be measured. The diversity of a sample is a combination of the number of species present in the sample (species "richness") and the degree to which all abundances in the sample are equal (species "evenness") [78].

There are a number of tools available for community profiling of metagenomic data. Some tools, such as MEGAN [33], infer community profiles based on the similarity of sequences in a sample to those of a sequence database. Each sequence in the database has a taxonomic lineage associated with it, so if a sequence in the sample has strong similarity to a sequence in the database it is likely to belong to the same taxonomic lineage. MetaPhlAn uses a similar approach, but uses a strongly constrained database of clade-specific marker genes [71]. This constrained search space allows for both unambiguous classifications (since each marker gene can only belong to a given clade), and shorter execution times.

2.8 Sequence assembly

Sequence assembly is the process of combining overlapping sequence reads into larger contigs. Assembly is a common process in metagenomic and genomic studies, as larger sequences allow for greater confidence in further downstream analyses (e.g., inferring homology of a newly sequenced gene via a database search). The assembly process is an essential part of single-genome studies. Abundance information is not relevant when working with a single genome, so the generated reads can be amplified via PCR and higher sequencing depth can be achieved. Higher sequencing depth greatly simplifies and enhances the assembly process as it reduces the chance of misassembly and so-called ‘chimeric’ contigs.

Modern assembler programs are usually graph-based or string-based. String-based assemblers such as SSAKE employ a greedy extension method, using structures such as prefix trees to extend contigs [79]. Despite the inherent simplicity in this method, string-based methods have not become as popular as the graph-based methods. Graph-based assemblers generally employ either a De Bruijn graph (DBG) method or overlap-layout consensus (OLC) method. De Bruijn graph assemblers, such as Velvet [84], use series of overlapping k-mers as nodes. Edges connect nodes that can overlap with each other. Then, a genomic sequence can be found by traversing the graph. OLC-based assemblers like MIRA [14], CAP [31], and Newbler [12] use three steps: calculate overlaps among all reads, use the overlaps to determine the layout of reads, and then determine a consensus sequence (e.g., assembled genomic sequence). A study by Li et al. [47] provides an in-depth comparison of the two major graph-based methods.

2.9 Read simulation

In order to evaluate various types of metagenomics software and/or techniques, it is generally required to supply these programs (e.g., assemblers) with sequencing data. It is generally not practical nor cost-effective to generate real data for which the results of subsequent analysis are known to test these programs. Also, if we generate ‘new’ data we have no idea how useful the results generated from a program will be. It is far more convenient to simulate sequencing in silico. For example, if we perform a sequencing simulation using previously sequenced genomes, then the most useful output generated by an assembler is that which is closest to the original genomes. Thus, simulated data improves our ability to determine whether or not a given metagenomics program or technique is well-suited to the tasks one would perform with ‘real’ data.

However, generating simulated read data is not as simple as just randomly selecting subsequences from a reference genome and then supplying that as input to a program. Given that reads often have sequencing errors, variable read lengths, and often nucleotide content biases, more sophisticated strategies are required when a user wishes to obtain useful results from simulated data. Additionally, if one were to just sample read lengths from a genome there would not only be a lack of sequencing errors but a lack of quality information which many programs use to determine whether or not a given base is erroneous.

Additionally, while sets of reads are often subjected to quality filtering to remove reads that are too short, too long, or too low quality, using realistic simulated data allows the user to test how exactly the quality filtering steps can affect one’s results and which parameters would be most appropriate. For example, filtering reads based on relative *k*mer abundance has been shown to negatively affect species composition in simulated metagenomic data, but have less serious consequences on single-genome data where species composition is irrelevant [36].

2.9.1 Current sequence-read simulator programs

There are currently a large number of sequencing simulator programs available to users, all of which take different approaches to generating data. In this section, we highlight the published features of five of the more popular simulators: 454sim, SimSeq, MetaSim, Grinder, and GemSim. An evaluation of the data generated by these programs is provided in Section 4.4.1.

2.9.2 Common features

All read simulator programs generate simulated data by performing at least two tasks: read sampling, and error model application. To sample reads from a given genome sequence $G = g_1 \cdots g_m$ of length m, the simulator randomly selects subsequences of length $l, 1 \leq l \leq m$ where l is some potential read length for a given sequencing technology. Thus, a very basic simulated read would be the sequence $g_i \dots g_{i+l-1}, 1 \leq i \leq m-l+1$. The simulator program's error model E then has some associated error rate probability $0 \leq e_i \leq 1$ for all positions i. If the simulator determines that an error occurs at position i, then nucleotide g_i (or g_{i+1}, for insertion errors) will differ from the genome in some way. This depends on the particular error model used by the simulator. The simulator in the following subsections all differ with respect to how they derive their error models, and the technologies for which error models they support. All sequencing simulator programs provide a command-line interface (CLI) for the user, while graphical-user interfaces (GUIs) are far less common.

2.9.3 454sim

454sim is a sequence simulator for three generations of 454/pyrosequencing technology. This program is an extension of the sequencing simulator Flowsim [6], and requires two steps to generate output in order to generate raw sequencing data in SFF format. First, a program called FragSim is supplied with a file of reference genome(s), the number of desired reads, and the lengths of the reads. FragSim then outputs uniform-length reads as a multi-FASTA file. Then 454sim is used to generate raw sequence data based on the output from FragSim and outputs an SFF file containing the raw sequence data. 454sim uses a normal distribution for modelling positive flows (i.e., one or more bases), and a log-normal distribution for modelling negative flows (i.e., noise flows) [53]. Additionally, degeneration models and models for deriving flow-peak values are included in 454sim [53]. The main advantages of 454sim are speed and multithreading capability. By default, the program uses 16 cores to generate sequence data, and is capable of generating 10,100 reads per second on an Intel Core i7 920 processor [53]. Thus, 1,000,000 reads can be generated in less than two minutes.

2.9.4 SimSeq

The purpose of SimSeq was to develop an Illumina read simulator based on the capabilities of Illumina-supplied software to model paired-end reads, with the developer's own position and reference-base-specific empirical error model trained on Illumina data. It was developed as part of Assemblathon 1, a competition which allowed for benchmark testing of 41 assemblers [20].

SimSeq requires at least 11 user-supplied parameters as input [72] and produces a SAM file as output. This output can easily be converted by the user into other formats such as FASTQ or FASTA, both of which are broadly accepted by many bioinformatics tools. Since SimSeq does not support any sort of abundance or diversity parameters, it is likely that this program was intended to simulate the sequencing of individual genomic sequences.

There are a few documented limitations of SimSeq [20, 73]. One problem is that the reads are randomly sequenced within chromosomes, but not randomly sequenced between chromosomes. This has the possibility of being a source of bias during assembly [73]. The most obvious limitation of SimSeq is that it cannot simulate Roche/454 data or any other non-Illumina platforms. It also does not appear to have an empirically derived error model (i.e., based on actual generated Illumina reads), but an error model based on errors typically introduced by paired-end sequencing. Another limitation of SimSeq is that the error rate is independent of the error of the previous position. There is a position-specific model available, but each position is still sampled independently of previous and subsequent positions. Additionally, there were plans to add in a feature where adapter sequences show up at the ends of reads when the fragments are too short, but this functionality is not present [73]. There have not been any updates to SimSeq posted online since 2012, so it is possible that these limitations will not be addressed in the near future [72].

2.9.5 MetaSim

MetaSim is one of the oldest sequencing simulators, and the first metagenomics sequencing simulator [63]. Unlike SimSeq and 454sim, tasks involving MetaSim can be performed in a GUI, which is useful for organizing multiple different simulations involving different reference genomes. MetaSim also has the ability to evolve selected genomes using various mutation models, allowing for the simulation of many closely related microbe strains in a population.

MetaSim requires a set of known reference genome sequences and an abundance profile as input. The GUI supplies a default abundance profile in which all genomes are equally abundant, but this file can be manually configured to support the simulation of sequencing more complex environments. The error profiles for sequencing are all supplied by default, but these can also be manually configured and MetaSim supports user-supplied error profiles as well. After sequencing, MetaSim provides a detailed log of the simulation, and a compressed multi-FASTA file containing the reads from the simulation.

MetaSim can simulate Sanger, 454, and Illumina sequencing. It is also able to model paired-end sequencing by extracting subsequences of a given length and standard deviation (with a normal distribution) from the selected genomes. Each genome sequence is assigned a weight which is used to generate a frequency for each genome. The ends of clones are then the basis for read or paired-end sampling. To simulate Sanger sequencing, MetaSim uses a linearly-increasing error rate with fixed values for indels and substitutions. For 454 sequencing, a normal distribution is used for positive flows and a log-normal distribution for negative flows (much like 454sim). Unlike SimSeq, MetaSim uses empirically-derived parameters for its error model of Illumina sequencing.

2.9.6 Grinder

Grinder is a sequence simulator that can generate genomic, metagenomic, transcriptomic, and metatranscriptomic sequence reads from reference genomes. Like other sequence simulators, it can simulate Sanger, 454, and Illumina data with support for paired-end reads and variable insertion (insert) sizes. It is mentioned in the original Grinder publication that there will be support for Ion Torrent sequencing when an error profile becomes available [3]. Grinder has a number of parameters that can be specified. The only parameter that is absolutely necessary is the reference genome file, but there are a significant number of optional parameters [4].

When simulating Sanger sequencing, Grinder uses an error rate which increases along the length of each read. For Illumina sequencing it uses a fourth-degree polynomial model, and for 454 sequencing it supports multiple homopolymer error models, one of which is the model described by Balzer et al., the same model used by Flowsim / 454sim [53, 6].

Grinder was the first published amplicon sequence simulator, and therefore can generate amplicons for genes of interest by simulating the PCR process [3]. For Grinder to simulate read sequencing for amplicon libraries, the program extracts full-length amplicons from the reference sequences based on user-supplied primer sequences, rather than uniformly sampling reads across the entire genome sequence. This PCR simulation also provides the opportunity to generate natural biases (e.g., due to variation in copy number) and biological artifacts.

2.9.7 GemSIM

GemSIM (short for General Error-Model Simulator) is the most recently published program in this section, and takes a unique approach to sequencing simulation. Rather than having built-in sequencing models for specific technologies with tuneable parameters, GemSIM takes a sample of user-supplied reads as input that have been aligned to some reference genome and uses the alignment information to build its error model [55]. That is, GemSIM can build error models with no knowledge of the specific technology used to generate the actual reads. GemSIM stores the following information from the alignment file when building an error model [55]:

1. the nucleotide type and position within the current read,
2. whether or not the current base is a mismatch or correct alignment,
3. whether or not there are indels following the current position,
4. the preceding three bases in the read,
5. the following base in the read,
6. the quality scores for the correctly matched, mismatched, and inserted bases.

That is, GemSIM uses 'sequence-context words' to determine the possible errors and quality scores to assign to simulated nucleotides. For example, if the word `ATTGC` in the simulated reads aligns to the word `ATCGC` in the genome, GemSIM would build a model where `ATCGC` is likely to have a substitution error at the third position, whereas a word that aligns perfectly to a reference genome would not have any errors in the simulated data.

Chapter 3
Research Goals

There are many bioinformatics tools for analyzing (meta)genomic sequence data. Evaluating these tools with raw data can be difficult if there is not a known "answer" for the data. This is particularly true for metagenomic studies, where experiments can involve sequencing thousands of novel organisms with little or no significant sequence similarity to other organisms in current databases. With artificial data it is possible to evaluate these bioinformatics tools against known or previously established results. However, a known problem with artificial sequence data, especially in the realm of metagenomics, is that it is not as "messy" as real data. Thus, this thesis has three goals. The first goal is to design and analyze a program, BEAR (Better Emulation for Artificial Reads), that can be used to generate realistic, simulated data that approaches the complexity or "messiness" of real data. The second goal is to analyze BEAR by comparing it to other sequencing simulator programs and evaluate all of these programs by the degree to which they can model real data. These goals are described in Section 3.1 and 3.2, respectively. The third goal of the thesis is to demonstrate how BEAR can be used to evaluate various programs for sequence analysis, focusing on the evaluation of assembly programs (Section 3.3.1) and metagenomic read classification tools (Section 3.3.2).

3.1 Creating BEAR

The purpose of BEAR is to allow a user to generate realistic simulated genomic and metagenomic data with minimal parameter manipulation. For BEAR, we will adopt a machine learning approach in which as many parameters as possible can be inferred from a sample of real sequencing data. While other programs exist for generating simulated data, they either have a very large parameter space or require a high-quality alignment to a reference genome. It is usually not feasible to have a reference genome for metagenomics experiments due to the exceedingly large number of species in a sample, many of which may be completely novel organisms. Thus, the creation of BEAR was motivated by attempting to provide an alternative sequencing simulator; one in which the parameters can be inferred strictly from the data, free from external sources (e.g., reference genomes). The parameters BEAR is largely concerned with are discussed in the previous chapter: read length distributions (Section 2.2), quality score profiles (Section 2.3.1), GC bias (Section 2.4), and in the case of metagenomics, relative species abundance (Section 2.5). Specific details describing how BEAR emulates these characteristics are provided in Section 4.4. With these motivations in mind, we present the following specific goals we wish to achieve with the creation of BEAR:

1. Implement machine learning approaches to "learn" the characteristics of a given sample of NGS reads. This will include implementing and comparing different types of Markov chains for generating quality scores.

2. Understand and evaluate how the various components of BEAR behave in order to determine how well BEAR can emulate artificial data.

3. Implement BEAR as a series of discrete Perl and Python scripts. When large amounts of data are processed and generated, there is always a possibility that a program can be interrupted. Breaking BEAR into discrete stages, rather than being one large program, minimizes the risk and loss of resources if a process were to be interrupted.

4. Introduce a reference genome-free approach to deriving error models. This may be accomplished by using a clustering tool such as DRISEE [41]. This type of approach would allow BEAR to generate models for datasets with reads from novel organisms (e.g., metagenomic reads), which is a problematic situation for reference genome-dependent tools like GemSIM [55].

5. Develop two methods for generating abundance profiles: a parametric method and an empirical method. The parametric method will be a faster, simpler approach where abundances are determined by power functions corresponding to species evenness outlined in previous work [56]. However, microbial communities can be complex and aren't necessarily found in nature to have abundances that fit to some smooth mathematical function. Thus, we will provide the user with the option of generating abundance files by using database searches. That is, the reads in a metagenomic dataset are aligned to a microbial protein database, and the organisms that match more often to the set of reads are assumed to be more abundant in the community. This allows for complex, empirically-derived abundance files to be generated, a feature absent from other metagenomic sequencing simulator programs.

6. Develop an algorithm for emulating the GC profile of a given sample of reads. This task may require thorough pre-processing of genomic sequences, so it is a goal to develop a scalable approach so that, for example, a computer with a large number of processors would be able to finish this task quickly.

We also present the following non-goals:

1. It is not currently a goal to have a graphical user interface (GUI) for BEAR. BEAR will be executed using a command-line interface for the time being, with the development of a GUI left for future work.

2. It is not a goal of this thesis to provide a rigorous performance analysis (e.g., memory usage, execution time) of BEAR and the other various sequencing simulator programs, although we will ensure that BEAR's computational performance is practical for a modern workstation-class computer. Rather, the thesis will focus more on the quality and accuracy of data generated. Given that some of these programs (e.g., BEAR and GemSIM) use machine learning approaches it is expected that they will be slower than, for example, a program that uses pre-set models. This thesis will focus more on the effectiveness of each program's ability to generate realistic data.

3. There are no explicit performance or resource utilization goals for BEAR beyond scalability for GC profile emulation.

4. Metagenomics experiments typically utilize biological replicates (i.e., the analysis of multiple samples across multiple conditions) and technical replicates (i.e., the analysis of a single sample across multiple conditions) to measure variation in the results of analyses [**?**]. However, some of the methods analyzed in this thesis (namely the MIRA assembler and the MEGAN community profiling tool) can have runtimes on the order of days to complete. Given the volume of data generated for each simulated data sample, the number of sequence analysis tools we seek to evaluate, and time constraints, we will not generate replicates (technical or biological) for the work in this thesis.

Furthermore, we present the following limitations of this research:

1. BEAR will only be evaluated on its ability to emulate Ion Torrent, Illumina, and pyrosequencing sequence data. Data from emerging technologies such as SMRT sequencing will not be used.

2. BEAR will make the assumption that the error models and quality scores apply equally to all reads in a sample (an assumption made by other programs as well). That is, there will only be one Markov chain and one error model per sample and not, for example, an error model for "good reads" and an error model for "bad reads".

3. BEAR will be explicitly designed only to emulate shotgun reads and not other types of sequencing experiments (e.g., amplicon sequencing).

3.2 Analyzing data from sequencing simulator programs

One fault with many existing sequencing simulator programs is that their associated publications lack comparisons to other existing programs [55, 53, 63]. When papers do feature comparisons of the programs, the analysis often ends at a high-level qualitative comparison [3]. While comparisons of this nature can be useful, they lack detail with respect to demonstrating how the simulated data actually compares to real data. Thus, one goal of this thesis will be to evaluate the data generated by all of the simulator programs listed in Section 2.9 based on their abilities to simulate characteristics of real sequencing data, and compare them to BEAR. This includes generating simulated Ion Torrent, Illumina, and pyrosequencing data and comparing the read length distributions, quality score profiles, and error rates of these programs.

3.3 Analyzing programs using simulated data

Although the purpose of generating simulated sequence data is to evaluate sequence analysis programs, some papers introducing sequencing simulators fail to conduct such analyses [55, 53]. Although these papers adequately describe the characteristics of data they were able to generate, it is imperative that subsequent analyses using the simulated data are performed to verify that they work as intended. To this end, we will compare various sequence analysis programs by using the simulated data from BEAR and other simulator programs that were found to closely match the characteristics of real data. In particular, we will focus on comparing two genome assembly programs and two metagenomics assembly programs.

3.3.1 Using simulated data to evaluate assembly programs

This particular goal is motivated by frequent observations from various sequencing simulator and genome assembly programs. Genome assembly is a common task for many sequencing experiments. The assembly of short reads into larger, contiguous sequences (contigs) can, for example, allow for the data to be classified to a given organism or having a family of genes. Given how ubiquitous the task of assembly is, it is perhaps surprising that few tests have been conducted to assess how useful simulated data is for evaluating assembly programs. Of all the programs mentioned in Section 2.7, only SimSeq has any published assembly analyses using simulated data[20]. Thus, one goal of this thesis will be to determine if an assembly of simulated data is at all similar to an assembly of the real data, and how well sequencing simulator programs can predict whether one assembler is better than the other. Furthermore, another goal of this analysis will be to measure the effects of sequencing errors and GC profiles on genome assembly by using the methods designed in Section 3.1. In the interest of simplicity, this analysis comes with the caveat that only two sequence assemblers will be compared (Velvet and MIRA), and only the top three sequencing simulator programs from Section 3.2 will be used to generate simulated data.

3.3.2 Using simulated data to evaluate metagenomics classification programs

A common task for metagenomics experiments is to attempt to classify reads to organisms or taxonomies in order to characterize the environment. Some metagenomic sequencing simulators perform classification experiments with their simulated data [63, 3], but these studies are generally proof-of-concept rather than attempts to compare different classification tools. Thus, a goal of this thesis will be to compare the abilities of the metagenomic sequence classification tools MEGAN and MetaPhlAn [33, 71] and measure the effects of GC bias on metagenomic classification (a feature that, to our knowledge, has yet to be evaluated). Since metagenomic classification can be a very time-consuming process (especially with tools that align millions of reads to extremely large databases like the sequence identity-based approach included with BEAR), we will only use simulated metagenomic data generated by BEAR to evaluate the two classification tools.

Chapter 4

Data and Methodology

This section describes the data and methodology used to achieve the three main goals of the thesis. The real data used for training and evaluating BEAR are described in Sections 4.1. A high-level overview of the design of BEAR is provided in Section 4.2, while details of the implementation are provided in Section 4.3. The methodology for evaluating the data produced by various sequencing simulator programs is provided in Section 4.4.1. Finally, the methodologies for evaluating assembly programs and metagenomic classification tools are outlined in Sections 4.4.2 and 4.4.3.

4.1 Real data

Four sets of reads from commonly used sequencing platforms were used to evaluate BEAR and other sequencing simulator programs. DNA was extracted from sediment samples collected from Black Lake, Saskatchewan at a depth of 0-2cm using the MoBio PowerLyzer PowerSoil DNA Isolation Kit. An Ion Torrent Personal Genome Machine with an Ion 318 chip was then used to sequence a dataset consisting of 377,630 raw metagenomic reads. Three datasets consisting of non-metagenomic reads were also used: 689,365 reads from the *E. coli* DH10B genome using a Personal Genome Machine with an Ion 318 chip; 122,737 reads from from the *L. rhamnosus* ATCC 8530 genome using a Roche 454 Genome Sequence FLX platform and 100,000 transcriptomic reads from the *P. claussenii* ATCC BAA-344 genome using an Illumina Genome Analyzer IIx. The two Ion Torrent datasets were data made available to us from colleagues at the University of Saskatchewan, while the latter two datasets were obtained from Pittet et al. [SRA: SRX216314] [57, 58]. The sequence simulators were then evaluated by how closely they were able to emulate the characteristics of the training datasets.

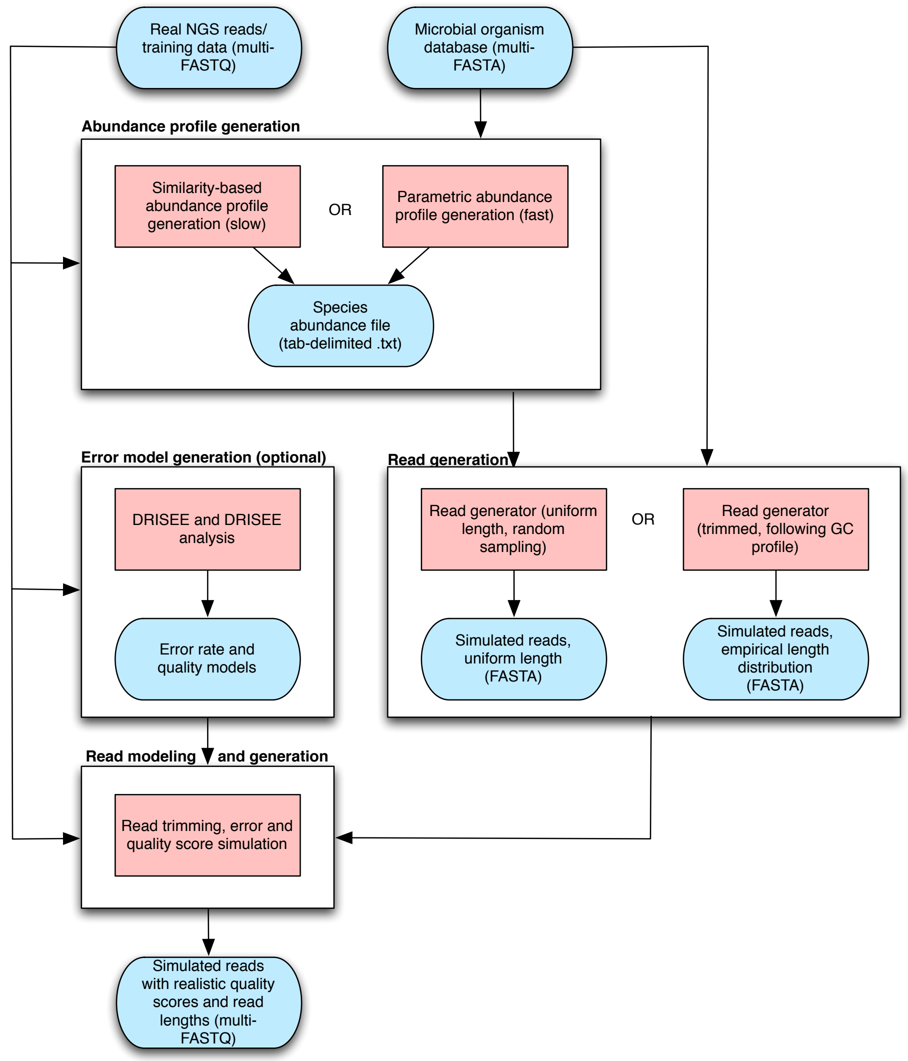
4.2 Design

An overview of the BEAR workflow can be found in Figure 4.1. At a high level, the purpose of BEAR is to take in a set of sequencing reads in FASTQ format and a set of known genome sequences and output a new set of simulated sequencing reads having the characteristics of a user-supplied dataset of sequencing reads. As shown in Figure 4.1, BEAR accomplishes this over the course of four steps: abundance profile generation, error model generation, uniform read generation, and read trimming and error incorporation.

4.2.1 Abundance profile generation

Abundance profile files are necessary input for existing metagenomic sequencing simulators. Despite it being an input for most metagenomic simulators, BEAR can optionally create abundance profiles and therefore is an output of this stage of the pipeline. A common format for such files is that used by Grinder and GemSIM, which is tab-delimited text where the first column is a genome identifier, and the second column is the relative abundance (a number between 0 and 1) of that genome in the simulated community. BEAR not only accepts abundance profiles in this format, but also provides users with the resources to generate them for any number of organisms. Of the other programs mentioned in Section 2.7, only MetaSim provides users with the means to create abundance files. However, the abundance values and organisms have to be manually selected or entered, which can be time consuming for simulating complex environments. While BEAR can accept manually-curated abundance files as input, we also wanted to allow users to automatically generate complex abundance files. Furthermore, BEAR is designed to support two methods for generating abundance files:

- **Power function-based abundance profile generation (fast, parametric)**: This module of BEAR requires two parameters: a set of genome sequences for which to assign relative abundances, and a type of community evenness. This abundance values for the entire simulated community are derived from one of three user-specified power functions of the form $abundance(x) = ax^b$, where a is a constant representing the highest abundance value in the entire community, x is the abundance rank of a given organism (e.g., $x = 1$ for the most abundant organism, $x = 2$ for second-most abundant organism, etc.), and b is a constant associated with the evenness of the community. BEAR assumes that the order of genomes present in the user-supplied file is the rank-order for deriving abundances, and that $abundance(1) \geq abundance(2) \geq \cdots abundance(n)$ over the range $[0, 1]$ such that $\sum\limits_{1 \leq x \leq n} abundance(x) = 1$. BEAR has three defined power functions for communities with differing degrees of evenness:
    - **Low evenness**: $abundance(x) = 31.4x^{-1.287}$
    - **Medium evenness**: $abundance(x) : 21.23x^{-1.287}$
    - **High evenness**: $abundance(x) : 2.01x^{-0.233}$



Real NGS reads/ training data (multi-FASTQ)
Microbial organism database (multi-FASTA)
Abundance profile generation
Similarity-based abundance profile generation (slow)
OR
Parametric abundance profile generation (fast)
Species abundance file (tab-delimited .txt)
Error model generation (optional)
DRISEE and DRISEE analysis
Error rate and quality models
Read generation
Read generator (uniform length, random sampling)
OR
Read generator (trimmed, following GC profile)
Simulated reads, uniform length (FASTA)
Simulated reads, empirical length distribution (FASTA)
Read modeling and generation
Read trimming, error and quality score simulation
Simulated reads with realistic quality scores and read lengths (multi-FASTQ)

Figure 4.1: BEAR workflow. There are four major stages of using BEAR: error model generation, abundance profile generation, uniform read generation, and read modelling and generation. Blue rounded rectangles represent data files, red rectangles represent processes. Incoming and outgoing arrows represent input and output to and from processes, respectively.

A community with“low” represents an environment with few dominant species, while a community with “high” evenness has no dominant species.

- **Sequence identity-based abundance profile generation (slow, empirical)**: This method derives abundance values by analyzing the similarity of sequences in a set of input reads to sequences in a protein database where each sequence in the database has an established taxonomic lineage. When the set of input reads are aligned to the protein sequences in the database, the protein sequences with significant sequence identity are mapped to their associated taxonomic lineages, and a lowest-common-ancestor algorithm is used to determine the level of classification (e.g., phylum, genus, species, etc.). The resulting classifications (taxonomic lineages) are aggregated to determine the relative abundances of each taxonomy in the input database. The abundances of each species-level classification are then used to create the abundance profile. Preliminary studies indicated that RAPSearch2 offered the best balance of running time and quality of database matches when compared to similar methods [85, 77], so BEAR assumes that this is used as the database search tool. However, BEAR also provides instructions for converting results from other sequence identity-based search tools to the same format as the RAPSearch2 output.

The output of this stage is a tab-delimited abundance file where the first column is a genome identifier and the second column is a relative abundance value between 0 and 1, such that the sum of all relative abundance values equals 1. This format is accepted by other metagenomic sequencing simulators (GemSIM, Grinder), so a user could use the output from this stage as input to a different simulator as well.

4.2.2 Error model generation

BEAR is designed to be sequencing platform-agnostic and not require alignments to reference genomes. Thus, it is required to use methods that can be derived from the information contained in a multi-FASTQ file of input reads. With these restrictions, we can allow BEAR to derive two types of error models:

- **Quality-score based models (fast):** With this type of simple error model, the error rate at a given position within a read is derived directly from the quality score at that position. In this model, substitution and indel errors are assumed to occur at equal rates and nucleotide-specific error rates (e.g., transition/transversion rates) are assumed to be equal as well. In the case of an insertion, the quality score of the inserted nucleotide is the same as the quality score that preceded it.

- **DRISEE-derived error models (slow):** A more complex error model derived from clustering all duplicate reads in a sample of reads using DRISEE (Duplicate Read-Inferred Sequencing Error Estimator) [41]. This method allows for complex, nucleotide-specific error-rate models to be derived without the use of any reference genomes by clustering duplicate reads and calculating the error rate at each read position for each type of error and nucleotide. BEAR can also use the DRISEE output to derive models for quality scores associated with substitution and insertion errors.

4.2.3 Read generation

Default usage

In this stage, the abundance file generated in the first step and a set of genome sequences are provided as input by the user. BEAR also requires three additional parameters: the total number of reads, a flag indicating whether or not paired-end reads should be generated, the longest possible read (plus the insert length, if generating paired-end reads), and the longest potential read in the dataset. BEAR then generates a file in multi-FASTA format consisting of uniform-length reads and having the species composition specified in the abundance profile.

Alternative: GC profile emulation

Optionally, the user may instead generate reads with GC bias derived from the set of input reads. The parameters required from user are the same as those described for default read generation, with the addition of a ‘minimum read length’ parameter (to improve performance, see Section 4.3.7) and the multi-FASTQ sequence file. BEAR then performs a ‘biased sampling’ of the genome sequences, such that regions with higher or lower GC content are sampled based on the frequency of reads with that GC content occurring in the multi-FASTQ file. As indicated in Figure 4.2, preliminary examination of real data indicated that read length can influence GC content of reads, so the reads generated in this stage are not of uniform length.

4.2.4 Read trimming and error incorporation

The final stage, BEAR takes as input the output from the previous stages in addition to a user-supplied set of sequences. BEAR then builds a read-length distribution and quality score profile from the latter file. If the reads are all of uniform length, they are trimmed based on the read-length distribution of the real data and has quality scores generated for each nucleotide based on a combination of the quality profile, error models, and error-quality models.

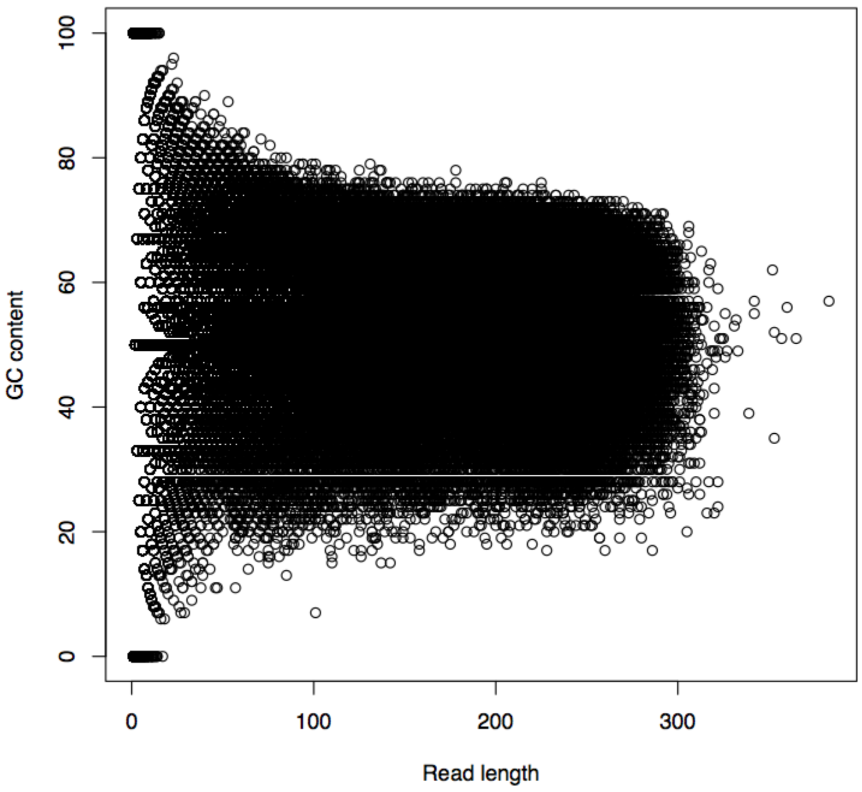


Figure 4.2: Plot of GC content and read length for Ion Torrent data. Shorter reads tend to have more extreme GC content values while longer reads tend towards GC contents in the range of 20-75% in this particular dataset.

4.3 Implementation

4.3.1 Abundance profile generation

Power function-based abundance profile generation

Abundance files were acquired from previous studies involving simulated metagenomic data [56, 54]. Both of these studies used three datasets of varying species complexities (low, medium, and high). Low evenness implies some dominant organisms in an environment, whereas high evenness implies no dominant organisms. These relative abundance values were plotted and we manually adjusted them to fit power law functions that would be representative of the species composition of an artificial metagenomic dataset. An example of a dataset pre- and post-adjustment can be found in Figure 4.3. All three adjusted abundance values and functions can be found in Figure 4.4. The result is three power functions of the form $y = ax^b$, where y is the relative abundance of an organism, x is the abundance rank of the organism, and a, b are constants that depend on the desired evenness of the community (low, medium, or high). The most dominant species is over 1000 times as present as the 118th most dominant organism in the low and medium evenness communities, whereas the most dominant species in the high evenness community would only be three times as present. Abundance files could then be generated by using Algorithm 1.

Sequence identity-based abundance profile generation

The methodology for RAPSearch2 [85] can be found in previous studies. BEAR makes the assumption that the user has used RAPSearch2 to align a set of metagenomic reads to proteins in the RefSeq database [60]. The output from RAPSearch2 is then stripped of all but three columns leaving a tab delimited file with the following data: the accession ID of the query sequence, the accession ID in the database, and the bit score indicating the quality of the match. Each entry of the RefSeq database has a corresponding file in genpept format, an example of which can be seen in Figure 4.5. In each genpept file are two key (for our purposes) pieces of information. First, there is an accession ID on the line beginning with `ACCESSION`. Second, there are a series of lines labelled `SOURCE ORGANISM`, indicating the species and associated taxonomy.

The first time this type of analysis is performed, BEAR processes the genpept sequences in the RefSeq database to compile a list of accession number / taxonomic lineage pairs. This list can then be used in subsequent executions of BEAR. BEAR then processes the RAPSearch2 search results and associates reads in the user-supplied dataset with a taxonomic lineage. The level of taxonomy of a given query (read) is calculated by compiling all the lineages for all search results that have a bit score within a certain neighbourhood of the highest one, finding the lowest common ancestor (LCA) among all those lineages and abundances. This

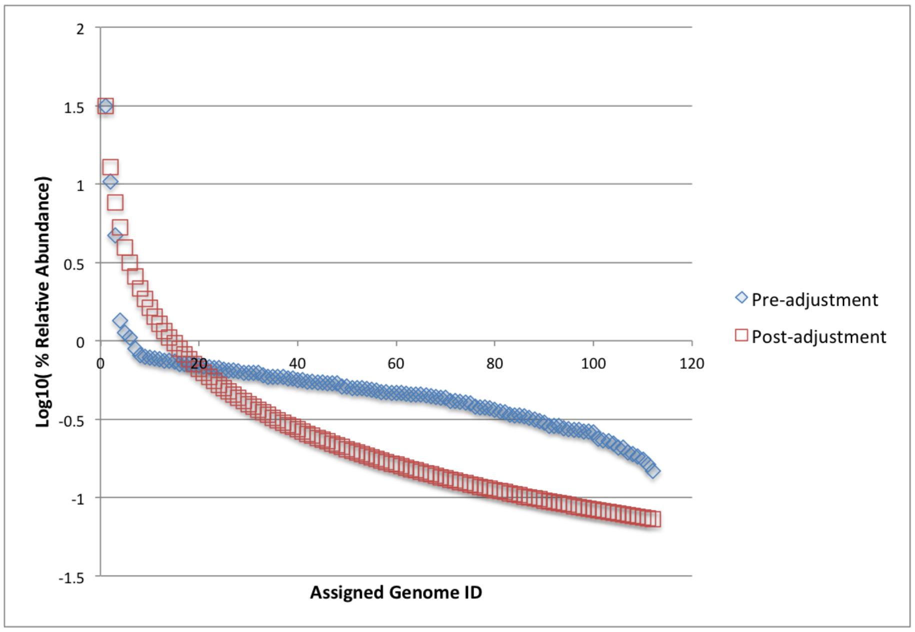


Figure 4.3: Example of adjustment of abundance values to allow for power law model derivation. This particular example represents a low-evenness model. Abundance values from previous work describing simulated metagenomic communities (labeled "Pre-adjustment") were adjusted to better fit power functions for generating abundance values (labeled "Post-adjustment").

```
Data: A list of genome sequence identifiers $G$, a type of community evenness $C$
Result: A tab delimited abundance file for a simulated community with any number of organisms
begin
    switch $C$ do
        case $low$
            $(const, exp)$ = (31.4, -1.287);
        case $medium$
            $(const, exp)$ = (21.2, -1.287);
        case $high$
            $(const, exp)$ = (2.01, -0.233);
        otherwise return error;
    endsw
    $i = 1$;
    for $g \in G$ do
        $abundance_i = const * i^{exp}$;
        $sum+ = abundance_i$;
        $i++$;
    end
    $i = 1$;
    for $g \in G$ do
        $abundance_i = \frac{abundance_i}{sum}$;
        print $g, abundance_i$;
        $i++$;
    end
end
```

Algorithm 1: Algorithm for generating abundance files with the power function-based method.

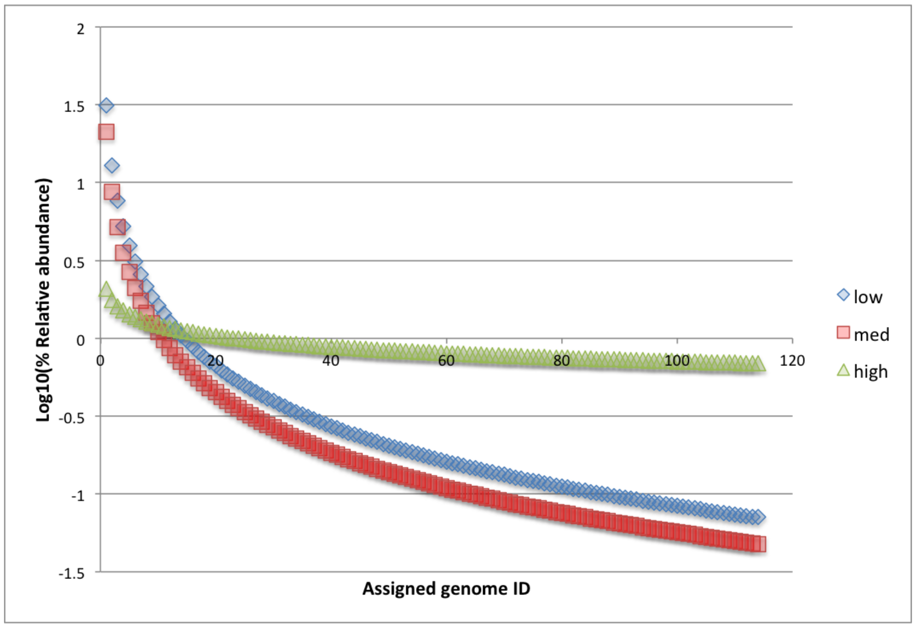


Figure 4.4: Abundance profiles derived from power regression. BEAR can generate three types of abundance profiles derived from power functions representing “low”, “medium”, or “high” evenness communities. Low evenness implies some dominant organisms in an environment, whereas high evenness implies no dominant organisms. Genome sequences were sorted by their relative abundance in descending order and then assigned IDs from 1 to n, where n is the number of genome sequences in the environment.

approach is very similar to that of MEGAN [33]. The result of processing the RAPSearch2 output is a tab-delimited file where the first column contains a taxonomic lineage and the second contains a relative abundance. This file is then subjected to Algorithm 2. This algorithm allows for the complex abundances to be mapped to any set of organisms the user prefers, without being restricted to the list of species-level hits found by RAPSearch2. However, this approach has the limitation of only being able to generate abundance values for a number of organisms up to the number of species-level classifications found by BEAR.

```
Data: A search summary file $F$ (processed RapSearch2 output), a list of genome sequences $G$
Result: A tab delimited abundance file for a simulated community with any number of organisms up
        to the number of species-level classifications in $F$.
begin
    Initialize list objects $S, T$;
    for $f \in F$ do
        if $f$ is classified to the species level then
            $S$.append($abundance_f$);

    end
    for $g \in G$ do
        $T$.append($g.header$);
    end
    for $i \in range(1 \dots length(T))$ do
        $sum += S(i)$
    end
    for $i \in range(1 \dots length(T))$ do
        $abundance = \frac{S(i)}{sum}$;
        print $T(i), abundance$;
    end
end
```

Algorithm 2: Algorithm for generating abundance files using processing RAPSearch2 output.

4.3.2 Error rate emulation

DRISEE-based error models

In order to generate error rate models, BEAR uses a modified version of DRISEE which, at its core, uses the uclust algorithm to cluster or ‘bin’ artifactual duplicate reads [41, 21]. The normal version of DRISEE removes the files detailing the clusters after running, but the modified version used by BEAR saves the cluster files and uses this information to derive more models in subsequent steps. BEAR and DRISEE classify “artifactually duplicate clusters” as any cluster of reads with identical 50bp prefixes (long reads) or 30bp (short reads). After DRISEE clusters the reads in a dataset, it generates a consensus sequence (i.e., a sequence using the most frequently observed nucleotides at each position) for each cluster. The reads in each cluster are compared to this consensus sequence to determine the mismatch/error rates. This is

```
LOCUS       AAB94881      842 aa                    BCT       05-JAN-1998
DEFINITION  DNA polymerase [Cenarchaeum symbiosum].
ACCESSION   AAB94881
PID         g2599106
VERSION     AAB94881.1  GI:2599106
DBSOURCE    locus AF028831 accession AF028831.1
KEYWORDS    .
SOURCE      Cenarchaeum symbiosum.
  ORGANISM  Cenarchaeum symbiosum
            Archaea; Crenarchaeota; Cenarchaeum.
REFERENCE   1  (residues 1 to 842)
  AUTHORS   Schleper,C., Swanson,R.V., Mathur,E.J. and DeLong,E.F.
  TITLE     Characterization of a DNA polymerase from the uncultivated
            psychrophilic archaeon Cenarchaeum symbiosum
  JOURNAL   J. Bacteriol. 179 (24), 7803-7811 (1997)
  MEDLINE   98062213
REFERENCE   2  (residues 1 to 842)
  AUTHORS   Schleper,C.M., Swanson,R.V., Mathur,E.J. and DeLong,E.F.
  TITLE     Direct Submission
  JOURNAL   Submitted (06-OCT-1997) Monterey Bay Aquarium Research Institute,
            PO Box 628, Moss Landing, CA 95039, USA
COMMENT     Method: conceptual translation supplied by author.
FEATURES             Location/Qualifiers
     source          1..842
                     /organism="Cenarchaeum symbiosum"
                     /db_xref="taxon:46770"
     Protein         <1..842
                     /product="DNA polymerase"
                     /name="archaeal family B DNA polymerase"
     CDS             1..842
                     /coded_by="AF028831.1:<1..2529"
                     /transl_table=11
ORIGIN
        1 vqdaveipps llvsatydsq agavvlkfye pesqkivhwt dntghkpycy trqppselge
       61 legredvlgt eqvmrhdlia dkdvpvtkit vadplaiggt nseksirnim dtwesdikyy
      121 enylydkslv vgryysvsgg kviphdmpis devklalksl lwdkvvdegm adrkefrefi
      181 agwadllnqp iprirrlsfd ievdseegri pdpkisdrrv tavgfaatdg lkqvfvlrsg
      241 aeegengvtp gvevvfydke admirdalsv igsypfvlty ngddfdmpym lnrarrlgvs
      301 dsdiplymmr dsatlrhgvh ldlyrtfsnr sfqlyafaak ytdyslnsvt kamlgegkvd
      361 ygvklgdltl yqtanycyhd arltlelstf gneilmdllv vtsriarmpi ddmsrmgvsq
      421 wirsllyyeh rqrnaliprr delegrsrev sndavikdkk frgglvvepe egihfdvtvm
      481 dfaslypsii kvrnlsyetv rcvhaeckkn tipdtnhwvc tknngltsmi igslrdlrvn
      541 yykslsksts iteeqrqqyt visqalkvvl nasygvmgae ifplyflpaa eattavgryi
      601 imqtishceq mgvrvlygdt dslfikdpee rqiheiveha kkehgvelev dkeyryvvls
      661 nrkknyfgvt ragkvdvkgl tgkkshtppf ikelfyslld ilsgvesede fesakmrisk
      721 aiaacgkrle erqiplvdla fnvmiskaps eyvktvpqhi raarllenar evkkgdiisy
      781 vkvmnktgvk pvemaragev dtskylefme stldqltssm gldfdeilgk pkqtgmeqff
      841 fk
//
```

Figure 4.5: An example of a sequence in genpept format.

performed for all clusters, and the results are tabulated and summarized to produce a tab-delimited output file. An abbreviation of this format can be seen in Figure 4.6. The five columns listing the estimated error rates are then subjected to exponential regression by BEAR to construct error models of the form $y = ae^{xb}$, where y is the error rate, a is the initial error rate, x is the current position in the read, and b is a factor that determines how quickly the error rate increases with x. BEAR produces five error models in total: a substitution error rate model for each nucleotide (A,T, G, and C), and a single indel rate model. BEAR also processes the clusters generated by DRISEE to determine the ratio of insertion/deletion errors at each position, nucleotide-specific insertion rates, and specific transition/transversion rates. The former is appended to a DRISEE output file with insertion and deletion columns, while the latter two are displayed in a matrix format. An example of this matrix format is provided in Figure 4.7

Quality score-based error models

This method calculates error rates for reads on-the-fly in the final step of the BEAR workflow. For a sequence-read of nucleotides $R = r_1 r_2 \ldots r_n$ we can calculate the error rate e_i for nucleotide $r_i, i \leq n$ based on the generated quality score q_i. Given that the probability of an error is directly related to the quality score by Equations 2.2 and 2.3, the total error rate at position i is $e_i = 10^{\frac{-q_i}{10}}$. This model makes the assumption that insertions, substitutions, and deletions are all equally likely. Additionally, for substitution errors, the probability of a base n_i being substituted by a base $m \in B = \{A, T, G, C\}$ s.t. $m \neq n_i$ is equal for all values of $m \neq n_i$. For example, if $n_i = A$, then $P(A \rightarrow T|substitution) = P(A \rightarrow C|substitution) = P(A \rightarrow G|substitution) = 0.\overline{3}$. Similarly for insertion errors, if $n_i = A$ then $P(A \rightarrow AA|insertion) = P(A \rightarrow AT|insertion) = P(A \rightarrow AC|insertion) = P(A \rightarrow AG|insertion) = 0.25$.

4.3.3 Error-quality modelling

DRISEE-based error-quality models

The stages of generating error-quality models are shown in Figure 4.10. The clusters generated by DRISEE/uclust to create the error rate models with the original sequence data file are supplied as input to a script that processes these clusters in order to find the quality scores associated with sequencing errors. As demonstrated in Figure 4.8, the format used by uclust for displaying sequence cluster information does not provide any quality score information. However, it does provide a sequence identifier. This identifier is used to map the cluster to FASTQ sequences such that the quality scores can be retrieved. The 8th column of the uclust format is called a CIGAR alignment string, consisting of integers and the characters M (match), I (insertion), and D (deletion). The integers preceding a M, I, or D indicate the number of consecutive bases for which the charac-

```
#       A        T        C        G        X
...
150 0.012902 0.005161 0.038706 0.038706 0.049027
151 0.020828 0.015621 0.020828 0.018224 0.067691
152 0.007843 0.002614 0.023530 0.020916 0.060132
153 0.010522 0.015784 0.015784 0.042090 0.092071
154 0.050316 0.010593 0.015889 0.015889 0.050316
155 0.007980 0.015960 0.034581 0.066502 0.055861
...
```

Figure 4.6: Portion of a DRISEE output file. For formatting reasons, some descriptive text in the file has been removed, in addition to columns irrelevant to the functionality of BEAR The first column represents the position within the reads. The next four columns represent the substitution error rates (as a percentage) of each nucleotide (i.e., the rate at which the nucleotide is substituted for any other nucleotide). The final column is the combined indel rate.

```
#Insertion matrix
#format {A,T,G,C}x{A,T,G,C}
0.07          0.67          0.14          0.12
0.08          0.26          0.11          0.55
0.50          0.19          0.14          0.17
0.27          0.38          0.10          0.24

#Substitution matrix
#format {A,T,G,C}x{A,T,G,C}
0.00          0.24          0.35          0.40
0.21          0.00          0.45          0.34
0.29          0.36          0.00          0.35
0.31          0.33          0.35          0.00
```

Figure 4.7: Examples of insertion and substitution matrices generated by BEAR after processing DRISEE/uclust output.

```
S    0    129    *       *      *      *      *       M82KP:378:342   *
H    0    129    93.8    +      0      0      92MD24MD11M2I   M82KP:419:2009  M82KP:378:342
H    0    129    93.8    +      0      0      115MI13MD      M82KP:498:1100  M82KP:378:342
```

Figure 4.8: An example of a sequence cluster in Uclust format. Each cluster has exactly one seed sequence (indicated by an S in column 1), and any number of hit sequences (indicated by an H. The remaining columns describe the following characteristics: Column 2: Cluster number; Column 3: Sequence length or cluster size; Column 4: Percent identity; Column 5: Strand; Column 6: Alignment start of query sequence; Column 7: Alignment start of seed sequence; Column 8: Alignment (CIGAR) string; Column 9: Identifier for query sequence; Column 10: Identifier for seed sequence. For formatting reasons, some descriptive text in the file has been removed.

```
Query/hit sequence:  -GTACGTACGT

                      |||||  |||

Seed sequence:       CGTAGG--CGT

Alignment string:    DMMMMMIIMMM

Compressed alignment string: D5M2I3M
```

Figure 4.9: An example of alignment strings found in clusters in uclust format using the sequences GTACGTACGT and CGTAGGCGT to generate the compressed alignment string D5M2I3M.

teristic is true. An example detailing this feature is provided in Figure 4.9. For example, the alignment string D5M2I3M would indicate that a 'hit' sequence (denoted in uclust format with an H in column 1) aligns with a 'seed' sequence (denoted with an S) with a deletion error, followed by five aligning bases, 2 insertion errors, and another 3 aligning bases. It is worth noting that an M does not indicate identity, it just means that two bases have aligned. That is, a base denoted by M in the CIGAR string can be the result of a substitution error.

The algorithm for processing the uclust file is provided as Algorithm 3. An example of the output from this stage is provided in Figure 4.11. This output is then subjected to second-degree polynomial regression, generating five models of the form $a_n x^2 + b_n x + c_n$, where $n \in A, T, G, C, I$ with A, T, G, C being the substitution error-quality models for specific nucleotides and I representing the error-quality model for insertion errors.

4.3.4 Generating uniform-length reads

Generating a file of uniform-length single-end reads only requires two or three parameters: an abundance file A consisting of pairs of organisms and abundances $A = (a_1, o_1), (a_2, o_2), \ldots, (a_n, o_n)$, a list of genome sequences $G = g_1, \ldots, g_n$ corresponding to the organisms in A, the total number of reads T , and the maximum read length M. Generating paired-end reads requires two additional parameters: the mean insert length,

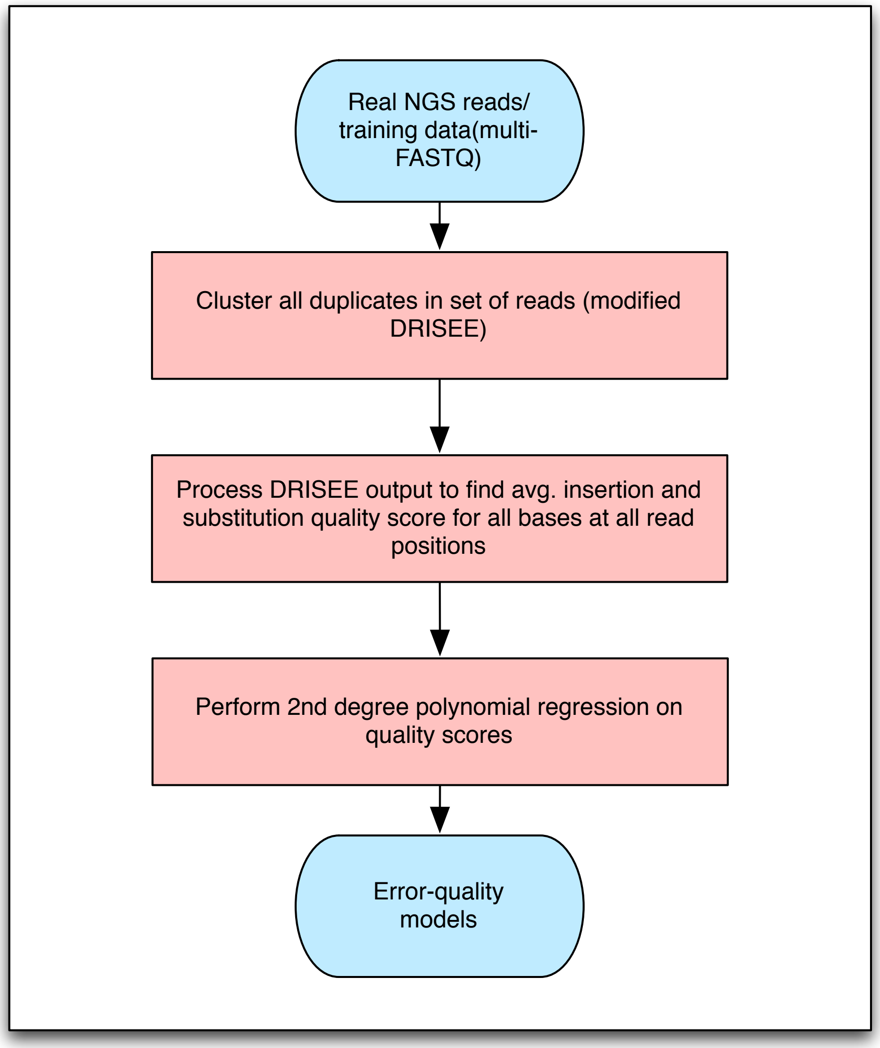

Real NGS reads/ training data(multi-FASTQ)
Cluster all duplicates in set of reads (modified DRISEE)
Process DRISEE output to find avg. insertion and substitution quality score for all bases at all read positions
Perform 2nd degree polynomial regression on quality scores
Error-quality models

Figure 4.10: Error quality pipeline.

```
Data: A list of clusters $U$ in uclust format, a list of reads $F$ in FASTQ format
Result: A tab-delimited file displaying the quality scores associated with sequencing errors.
begin
    Initialize $QualModel, SubMatrix$, and $InsMatrix$;
    for $u \in U$ do
        for $hit \in u.hits$ do
            $h.pos = s.pos = 0$;
            if $hit.identity \neq 100.0$ then
                for $e \in h.align\_string$ do // e has two elements:  a number (int) and type (M, I or D)
                    $n = e.num$;
                    $t = e.type$;
                    // track the current positions within the query (h) and seed (s) sequences
                    $h.subseq = h.seq[h.pos : h.pos + n]$;
                    $h.subqual = h.qual[h.pos : h.pos + n]$;
                    $s.subseq = s.seq[s.pos : s.pos + n]$;
                    $s.subqual = s.qual[s.pos : s.pos + n]$;
                    if $t == M$ then
                        for $i = 0; i < n; i++$ do
                            if $h.subseq[i] \neq s.subseq[i]$ then
                                // Add quality score of base to the list of other substitution error
                                   quality scores at the current position
                                $QualModel(h.subseq[i], i + s.pos).append(h.subqual)$;
                                // Increase substitution error counts for substitution error matrix
                                $SubMatrix[h.subseq[i]][s.subseq[i]] += 1$;
                            end
                        end
                        $h.pos += n$;
                        $s.pos += n$;
                    else if $t == D$ then
                        // Deletion errors don't have quality scores, nothing to store
                        $s.pos += n$;
                    else
                        $prev = h.seq[h.pos - 1]$;
                        for $i = 0; i < n; i++$ do
                            // Add quality score of base to the list of other insertion error
                               quality scores at the current position
                            $QualModel('I', i + h.pos).append(h.subqual)$;
                            // Increase insertion error counts for insertion error matrix
                            $InsMatrix[prev][h.subseq[i]] += 1$;
                            $prev = h.subseq[i]$;
                        end
                        $h.pos += n$;
                    end
                end
            end
        end
    end
    Calculate average quality score at every position for each error in $QualModel$;
    Convert values in $InsMatrix$ and $SubMatrix$ to percentages;
end
```

Algorithm 3: Algorithm for processing clusters in uclust format to generate the error-quality values, insertion error matrix, and substitution error matrix.

```
#    A     T     G     C     X


150 19.93 17.35 20.92 17.55 21.50

151 17.15 23.05 16.44 16.88 16.17

152 20.29 17.45 21.44 16.09 15.60

153 17.12 22.00 20.79 19.82 22.57

154 19.59 19.70 23.40 21.30 21.40

155 18.50 21.60 15.20 20.86 19.75
```

Figure 4.11: Portion of the output file from Algorithm 3. For formatting reasons, some descriptive text in the file has been removed. The first column represents the position within the reads. The next four columns represent the average quality score associated with the substitution errors of a specific nucleotide. The final column is the average quality score associated with an insertion error at that position.

and the standard deviation of the insert length. The algorithm for generating reads with these parameters is provided in Algorithm 4.

4.3.5 Read-length distribution emulation

To perform read distribution emulation, a file of uniform-length simulated reads (i.e., the output from the program described in Section 4.3.4) $S = s_1, s_2, \ldots, s_n$ and a file of real WGS reads $R = r_1, r_2, \ldots, r_m$ are supplied as input. BEAR calculates the length of all $r \in R$, and converts these to frequencies, resulting in an empirically-determined read length distribution. Thus, the probability of a simulated read $s_i, 1 \leq i \leq n$ having a read length $L, min \leq L \leq max$ can be found by sampling from the distribution $P(L = l)$, which is determined by the following formula in Iverson notation:

$$P(l) = \frac{\sum_{j=1}^{m} [length(r_j) = l]}{m} \tag{4.1}$$

Sampling from this distribution results in a length $l_i, 1 \leq i \leq n$ to which a simulated read $s_i \in S$ can be truncated (BEAR always truncates the rightmost nucleotides).

```
Data: An abundance file $A$, a list of genome sequences $G$, the total number of reads $T$, and a
      maximum read length $M$. Paired-end reads require an insert length $mean$ and $stdev$ as well.
Result: A list of uniform-length sequence reads having the community profile specified in the
        abundance file
begin
    Let $n$ be the number of organisms in the community;
    for $i \in range(1, n)$ do
        $NumReads = a_i * T$;
        for $j \in range(1, NumReads)$ do
            if single-end reads then
                $start = randInt(1, length(g_i) - M)$;
                print $g_i[start : start + M]$;
            else
                $insert = SampleNormDist(mean, stdev)$;
                $start = randInt(1, length(g_i) - (2M + insert))$;
                $read1 = g_i[start : start + M]$;
                $read2 = g_i[start + insert : start + insert + M]$;
                print $read1$;
                print $read2$;
            end
        end
    end
end
```

Algorithm 4: Algorithm for generating uniform-length single- or paired-end (meta)genomic reads in BEAR.

4.3.6 Quality score profile emulation

Input WGS reads are used as training data to create a quality score model for correct base calls based on position-dependent Markov chains. An example of using a Markov chain to model quality scores is provided in Figure 4.12. BEAR uses the models for error rate, quality-error, read length distribution (see previous subsections), and these Markov chains along with the uniform-length reads to generate the final variable-length artificial reads. In this thesis, we evaluate three types of Markov chains for generating quality scores: first order Markov chains, mean state first-order Markov chains, and second-order Markov chains. We define a read of length p to be a string of characters $S = s_1 ... s_p$ with an associated quality string $Q = q_1 ... q_p$. For each read, BEAR uses the training data to generate quality values $q_i, 1 \leq i \leq p$ based on one of the following:

1. **First-order Markov chain**: The quality score at a given position within a read influences the quality score at the next position. That is, q_i is conditional on q_{i-1} and position $i - 1$ or $P(q_i | q_{i-1}, i - 1)$. The Markov chain is trained by simply counting the quality scores present in the input set of real reads at each position of each read and then converting the counts to transition probabilities.

2. **Mean state first-order Markov chain**: The average quality score of five consecutive positions influences the quality score at the next position. In this case, q_i is conditional on the position $i - 1$,

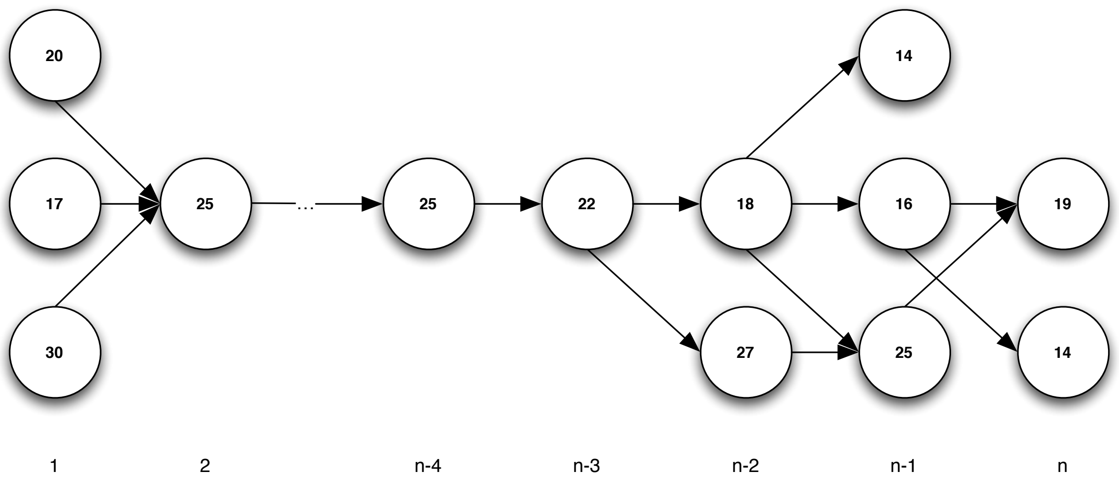


Figure 4.12: Example of Markov chain-based approach to generating quality scores. Integers within the nodes represent the quality score at a given position within the read. Each directed edge has an associated probability derived from the training data.

and the mean m_{i-1} of the previous quality values $q_k, q_{k+1}, \ldots, q_{i-1}$ where $k = max(1, i-5)$. That is, $P(q_i|m_{i-1}, i-1)$.

3. **Second-order Markov chain**: The quality score at a given position and the previous position influence the quality score at the next position. This model assumes that q_i is conditional on $q_{i-1}, i-1$, and q_{i-2}, represented by the conditional probability $P(q_i|q_{i-1}, q_{i-2}, i-1)$.

Thus, for a given position i within a quality string Q, we wish to find q_i by sampling from the appropriate conditional probability distribution for all values of i and in all simulated input reads.

These models are only for producing nucleotides that are correct base calls. If an erroneous nucleotide is to be generated, the error model overrides the predicted quality value for q_i. For example, in the case that the error rate model predicts a substitution error at position i, the Markov chain is not sampled and the quality-error model sets $q_i = a_{s_i} i^2 + b_{s_i} i + c_{s_i}$, where $a_{s_i}, b_{s_i}, c_{s_i}$ are the coefficients of the second-degree polynomial regression for the nucleotide s_i.

4.3.7 GC profile emulation

BEAR is able to emulate the GC profile present in a given set of WGS reads (as opposed to uniformly sampling reads from genomes) by extensively preprocessing the given genome to identify regions with certain GC contents. Given that length can influence GC content (i.e., short reads are far more likely to have 0% or

100% GC content than long reads), BEAR also considers read length when sampling from a genome. There are three discrete steps required for BEAR to emulate a GC profile: initiation, pre-processing, and sampling.

Initiation

When emulating a GC profile, BEAR considers all read lengths between a minimum length p and a maximum length q. This allows for read lengths that would later be filtered or removed to be ignored (i.e., extremely short or extremely long reads), saving time and computational resources. This user-specified constraint on read lengths should be taken into consideration when subjecting these simulated reads to any sort of quality control measures that consider read length to be a factor, as reads with lengths beyond these constraints will not be present as they would be in the original sample of reads. In terms of granularity, BEAR is only concerned with GC contents at the integer level. That is, the GC content of a read is rounded to the nearest whole number (e.g., a region with a GC content of 67.5% would be rounded to 68%). For longer reads, there will tend to be a large overlap in locations with a given length l and a GC content g. For example, the regions of a genome with a fitting the constraints $l = 301, g = 50$ may not be different than those fitting $l = 302, g = 50$. Thus, we can identify 'redundant' lengths between p and q to save time by only considering these "effectively unique" values. The algorithm for doing this is provided in Algorithm 5.

```
Data: A minimum read length $p$ and a maximum read length $q$.
Result: A list of effectively unique read lengths $L$, where $|L| \leq q - p$.
begin
    Initialize a list $L$;
    Let $x = 0.0$;
    for $i \in range(p, q)$ do
        $x += 1/i$;
        if $x \geq 0.01$ then
            // 0.01 is used because we are only concerned with GC content values between
               1 and 100
            $L.append(i)$;
            $x = 0.0$;
        end
    end
    Return $L$;
end
```

Algorithm 5: Algorithm for generating a list of effectively unique read lengths to use for pre-processing a genome for GC profile emulation.

Pre-processing

With the resulting output from Algorithm 5, we know that there is at most $|L|$ passes required to fully pre-process a given genome by its length and GC content (GC content can be calculated on-the-fly, so there is no need to do multiple passes for multiple GC content values). However, genome sequences are quite large (approximately. 4.5 million characters) and when $p - q$ is also large, processing a genome sequence can be time consuming. To alleviate this, BEAR uses a map-reduce multiprocessing approach to parallelize this task. This reduces the number of serial passes to approximately $\frac{|L|}{C}$, where C is the number of available CPUs.

In the *map* stage, a pool of C processes are spawned for genome lengths $q, q+1, \ldots, q+c$. The length $q_i, 1 \leq i \leq p-q$ associated with the process C_i is passed into Algorithm 6, where locations of all subsequences with length q_i and GC content $g_k, 0 \leq k \leq 100$ are identified and stored in a dictionary of lists d. When this is repeated $p-q$ times, the result is a set of dictionaries $d_1, \ldots, d_{p-q}$. For the *reduce* stage, these dictionaries are then merged into one dictionary D. From D, it is now possible to sample reads based on their GC content and length.

```
Data: A genome sequence $s$ and a sequence length $l$
Result: A dictionary of lists $d_l$ containing the locations of all $l$-length subsequences with GC contents
        $g_k, 0 \leq k \leq 100$
begin
    Initialize dictionary of empty lists $d_l$;
    for $i \in range(0, length(s) - l)$ do
        $g = GCContent(s[i : i + l])$;
        $d_l[(l, g)].append(i)$;
    end
    Return $d_l$;
end
```
Algorithm 6: Algorithm for preprocessing genome by length and GC content for a given length l.

Sampling

After processing the genome, reads can now be sampled by accessing D. Since D is indexed by keys of tuples $t = (l, g), q \leq l \leq p, 0 \leq g \leq 100$, we can sample a tuple t' from the original set of reads and then access a random member of the list $D[t']$. To prevent oversampling (e.g., a list $D[(l, g)]$ where $|D[(l, g)]| = 1$ could be sampled far more often than neighbouring list $D[(l + 1, g)]$, BEAR iterates through neighbouring lists by relaxing both parameters (length and GC content) by a value of 1 per iteration until a sufficient number of candidate sequences H has been obtained. This process is demonstrated in Algorithm 7.

```
Data: Total number of reads $R$, a genome sequence $g$, a candidate sequence threshold $H$, dictionary
      of lists $D$, a set $T$ of length/GC tuples from a file of real WGS reads
Result: A list of sequences emulating the GC bias present in the file of real WGS reads
begin
    for $i \in range(1, R)$ do
        Select random tuple $t \in T$;
        $candidates = D[t]$;
        while $|candidates| \leq H$ do
            Set $candidates$ to be an empty list;
            $k = 1$;
            for $j \in range(-k, k + 1)$ do
                for $l \in range(-k, k + 1)$ do
                    $t' = (t.length + j, t.gc + l)$;
                    if $t' \in D$ then
                        $candidates.append(D[t'])$;
                    end
                end
            end
            $k+ = 1$;
        end
        $start = random(candidates)$;
        print $g[start : start + t.length]$;
    end
end
```

Algorithm 7: Algorithm for generating reads with GC bias from a processed genome sequence.

4.4 Evaluation

4.4.1 Comparison of sequence simulator programs

We compared BEAR to five sequencing simulator programs based on their ability to emulate the characteristics of actual sequence data obtained from Ion Torrent, 454, and Illumina sequencers. When determining the input organism databases and abundance files, we used the specific genomes and relative abundance values listed in previous work with simulated metagenomic data [56]. The programs that do not support abundance files were supplied with just the database of genome sequences. For each of the tested programs, parameters were chosen that would generate the read length and quality score distributions that most closely matched those of the actual test data.

4.4.2 Evaluating assembly programs

To evaluate the effect of different artificial read generation programs on (downstream) assembler programs, we generated four simulated genomic datasets containing simulated Ion Torrent data: two from BEAR (one with GC bias, one without), one from Grinder, one from GemSIM and compared the results to an assembly of real WGS data.

Each simulated Ion Torrent dataset contained reads sampled from the *E. coli* DH10B genome (reference genome assembled with DNAStar's SeqMan assembler) with the characteristics (read length distributions, quality profiles, error rates, GC bias) of the real Ion Torrent data used to sequence the genome. Reads in both real and simulated datasets were then subjected to quality control, by first removing all reads with lengths not within one standard deviation of the mean length and then truncating the remaining reads to the position where a sliding window detected an average quality score below 20. The simulated datasets were each assembled by both the MIRA and Velvet assembly programs [14, 84]. The contigs of each assembly were then aligned to the DH10B genome by using Bowtie2 [43]. The quality of each assembly was then measured by the following characteristics:

1. Total number of contigs.

2. Total assembly size (i.e., the sum of all lengths of all assembled contigs).

3. N50, the length of the smallest contig C in the sorted list of all contigs such that the sum of all contig lengths from the largest contig to C is at least 50% of the total assembly size.

4. Longest contig.

5. Average percent identity resulting from aligning the contigs to the original genome sequence. This is effectively a normalized edit distance, $(L - E)/L * 100\%$, where E is the edit distance and L is the length of the contig.

6. Total genome coverage of assembly.

It is worth noting that the goal of this analysis is not for the simulated data to result in a "better" assembly. Rather, it is far more useful for a simulator program to generate data that results in a *realistic* assembly. Thus, the "best" program would be one that results in an assembly with characteristics that most closely match those of the real data.

4.4.3 Evaluating metagenomics classification tools

One of the goals of this thesis is to determine if BEAR can be used to evaluate metagenomics classification tools. To this end, three simulated metagenomics datasets were generated using BEAR and the genomes in the NCBI genomes database (`http://www.ncbi.nlm.nih.gov/genome`) to test classification tools:

- two low evenness datasets, one with GC bias and one without,

- one smaller dataset with uniform abundances, with GC bias.

The first two datasets used all 2,062 complete bacterial genome sequences present in the NCBI genomes database. For the third dataset, the 120 genomes used to make simulated communities in previous studies [56, 54] were used by concatenating all genomes together and using this as a large, single 'input genome' to BEAR. All 3 datasets each consisted of 1,000,000 simulated Illumina reads (67bp) using the Illumina data mentioned in Section 4.1 as the sample input for training BEAR. These simulated datasets were used to analyze two different metagenomic classification tools: the sequence identity-based abundance profile methodology used by BEAR for generating complex abundance files and a similar classification tool, MetaPhlAn [71]. The resulting output of each classifier tool was an abundance profile, and these abundance profiles were compared to the original low-evenness power function used for generating the simulated data to determine the accuracy of each tool. Additionally, this methodology allowed us to observe the effects that GC bias can have on metagenomic classification.

Chapter 5

Results

This chapter presents the results and findings of this thesis. Section 5.1 provides an evaluation of simulated data generated by BEAR and a comparison of BEAR to other sequence-read simulator programs. This includes an evaluation of the data generated from each program by their read length distributions (Section 5.1.1) and quality score profiles (Section 5.1.2). The best-performing programs were then compared to real data based on their error models (Section 5.1.3) and, in the specific case of BEAR, error-quality models (Section 5.1.4). To determine the usefulness of BEAR in evaluating downstream sequence analysis, three small case studies end the chapter. The first is a basic quality control analysis, where the data generated by Grinder, GemSIM and BEAR were subjected to simple quality control methods and compared to real data (Section 5.2). The second study subjected the filtered data to assembly by two different assembler programs, where the assemblies were compared to real data to determine if the assembly of a simulated dataset was at all reflective of the assembly of a real dataset (Section 5.3). The effects of GC bias and sequencing errors on assembly are also presented (Section 5.3.1). The final study used three simulated metagenomic datasets (two with GC bias, one without) and two metagenomic classification tools to determine the ability of each tool to generate an accurate abundance profile of the simulated community, in addition to observing the effects of GC bias on metagenomic classification (Section 5.4).

5.1 Evaluation of BEAR and comparison of BEAR to other sequencing simulators

Results of attempts to simulate NGS data with a suite of artificial read generator programs are provided in this section. A summary of our findings for the read length distributions, errors, and quality profiles for each of the tested sequencing programs are presented in Table 5.1. In general, most programs were only able to generate reads with lengths following a degenerate distribution (454sim, SimSeq) or a normal distribution (Grinder, MetaSim). With respect to generating realistic quality profiles and error models, each program behaved differently. The parameters of 454sim were difficult to calibrate due to the lack of

Program	Read length distribution	Quality profiles	Errors
MetaSim	Uniform and Normal	Not generated	User-defined, parametric
SimSeq	Uniform	High quality for first 80bp, low quality after	User-defined, parametric
Grinder	Uniform and Normal	Binary; either "good" or "bad"	User-defined, parametric
454sim	Uniform	Highly sensitive to parameter settings	User-defined, parametric
GemSIM	Empirical	Empirical	Inferred from alignment to reference genome
BEAR	Empirical	Empirical for correct base calls, second-degree polynomial for errors	Inferred from performing regression analysis on DRISEE data

Table 5.1: Summary of characteristics of read-length distributions and quality profiles for BEAR and popular sequencing simulator programs.

documentation explaining how the parameters affect the generated data. SimSeq was able to generate profiles with high quality scores for the first 80bp and low/variable quality for the last 20bp. SimSeq's parameters do not appear to be empirically determined, but it has been used successfully for evaluating assemblies of Illumina data [9, 20]. Grinder was able to generate reads with error rates derived from uniform, linear, and polynomial functions, but was only capable of generating two possible quality values per run (a "good" quality value for correct bases and a "bad" quality value for errors), which is highly uncharacteristic of raw reads. MetaSim provided number of options for user-specified error parameters. Unfortunately, it did not support the generation of quality scores. GemSIM was able to generate reads from empirical read length and quality score distributions. However, it was only derive error rates and quality scores by aligning reads to a reference genome. That is, GemSIM required training on WGS reads from a single genome and therefore was unable to directly generate data having the error, quality, and read length characteristics of a given metagenomic sample.

While we had each program generate three different simulated metagenomic datasets (simLC, simMC, simHC), the results for all three sets were indistinguishable in terms of the features (read length distribution, quality profiles) that were used for evaluation. Consequently, only the low evenness simulated dataset is shown in the figures unless otherwise specified.

GemSIM anomaly

While evaluating the sequencing simulator programs, we observed a strange behaviour with the number of reads being generated by GemSIM. By running GemSIM with the identical parameters, we observed that GemSIM would generate a different number of reads with each execution. For example, we specified

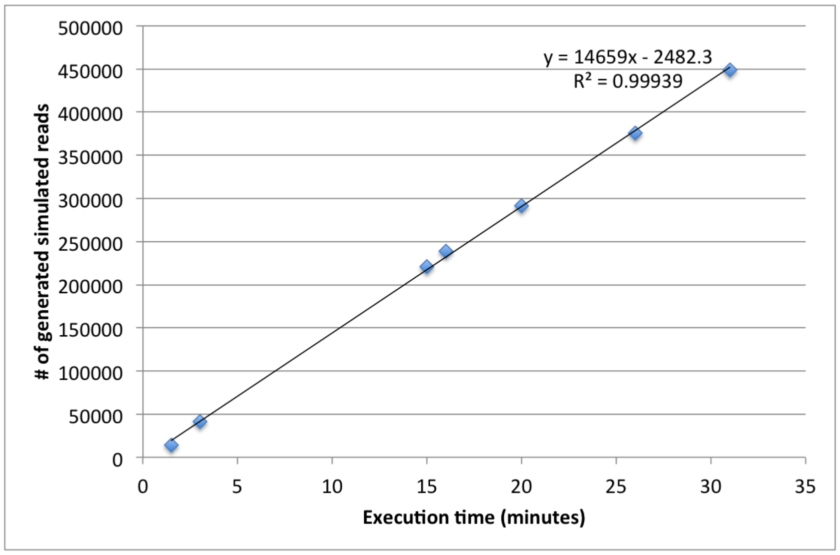
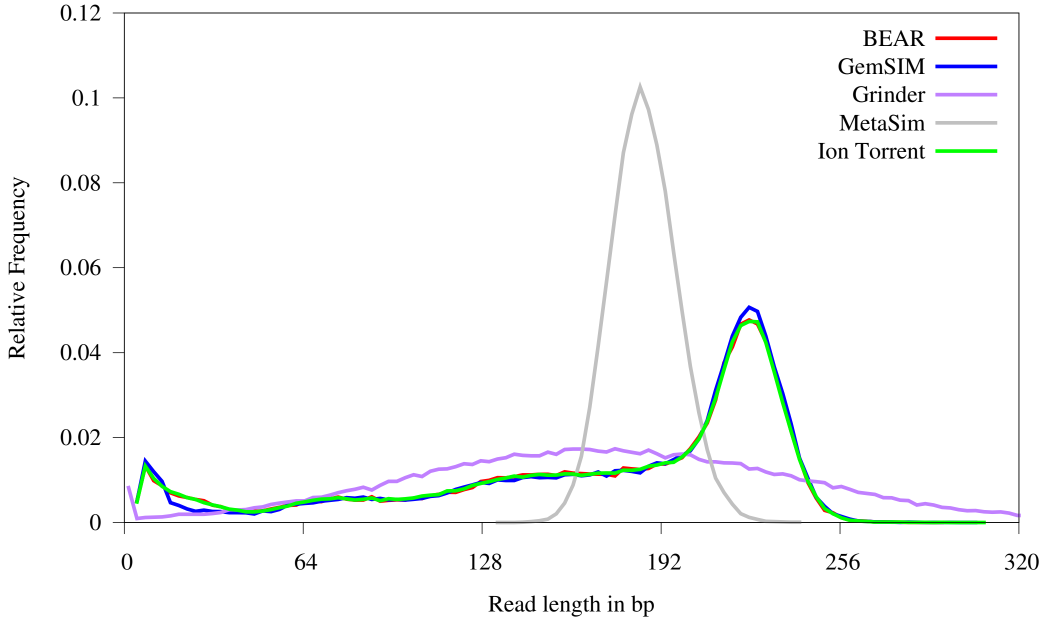


Figure 5.1: The linear relationship between real elapsed time and the number of simulated reads generated in a given GemSIM 'run' with identical user-supplied parameters.

to GemSIM that we wanted to generate 1,187,109 simulated reads and instead we obtained FASTQ files containing anywhere between 14,510 and 449,164 reads. Thus, we had to run GemSIM seven times with the exact same parameters in order to yield enough simulated reads. To our knowledge, this behaviour has not been documented anywhere. Figure 5.1 demonstrates the variance in the number of reads generated by a given GemSIM 'run' and the real time taken for that run to execute. There does appear to be a strong linear relationship between the number of reads generated and the time taken to generate those reads, but this behaviour is inconvenient for any user who wishes to generate a given, substantial amount of simulated data.

5.1.1 Read-length distributions

The read-length distribution of the real WGS training data was compared to the distributions generated by each sequencing simulator program, as demonstrated in Figure 5.2. From this figure, we observed that data obtained from actual NGS experiments was not necessarily simple enough to be characterized, for example, by supplying a mean and standard deviation of the read length distribution. GemSIM and BEAR were the only programs that closely modelled the Ion Torrent distribution. While the normally-distributed read lengths generated by Grinder and MetaSim model weren't as accurate as the read lengths generated by BEAR and GemSIM, they were far more accurate than those generated by 454sim and SimSeq. Over 80% of the reads generated by 454sim were 165bp, and no read lengths exceeded 175bp. SimSeq only generated reads 100bp in length. With respect to the 454 data, BEAR and Grinder matched the read length distribution far better than the other programs.



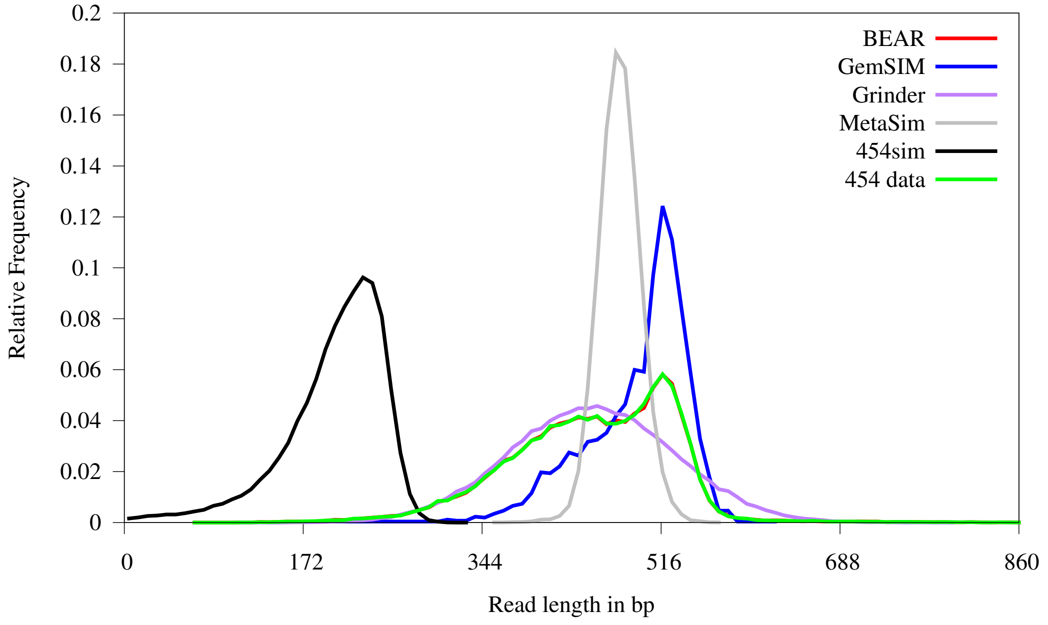


Figure 5.2: Comparison of read length distributions generated by metagenomics sequencing simulator programs. Top: Ion Torrent. Bottom: 454 data. When attempting to emulate these distribution, SimSeq only generated 100bp reads. Over 80% of the reads generated by 454sim were 165bp when emulating the Ion Torrent distribution, thus it is excluded from the top plot. Note that both panels BEAR closely matches the distributions of the real data. A panel for Illumina data is not shown since all real and simulated Illumina reads were 67bp in length.

5.1.2 Quality score profiles

Comparisons of quality profiles for all sequencing programs and WGS data can be seen in Figure 5.3. Similar to the read length distribution analysis, GemSIM and BEAR were the best of the tested simulator programs for generating data with the quality profile that most closely matched the real data. Near the end of the longer reads in the Ion Torrent and 454 data the base calls become quite noisy, leading to inconsistent quality values. While reads exceeding this length comprise a very small percentage of the data, it is worth noting that BEAR was able to generate noisy quality values after 250bp as well. Of SimSeq, Grinder, and 454sim, only SimSeq consistently produced non-constant quality scores. Unfortunately, it can only generate very short reads.

Comparison of different Markov chains for generating quality scores

Three different types of Markov chains were evaluated based on their abilities to generate quality profiles matching those of real Ion Torrent data. The types of Markov chains were as follows: a first-order Markov chain where the quality score of a given position was determined by the the average of the five previous quality scores, a first-order Markov chain where the quality score was determined by the previous quality score, and a second-order Markov chain where the quality score was determined by the previous two quality scores. The resulting quality profiles can be seen in Figure 5.4. As seen in this figure, the first-order Markov chain that used only the previous score appeared to better emulate the actual quality profile when compared the other two methods. The other first-order Markov chain was somewhat able to emulate the distribution, but the quality scores appeared to exhibit random behaviour at approximately the 280bp position rather than the 320bp position observed in the real data. The second-order Markov chain performed poorly, generating quality scores far lower than the real data for the first 250bp and higher than the real data for the last 130bp. For example, the second-order Markov chain generated quality scores with a median score below 20 by the 160bp position, whereas the median quality score of the real data did not fall below 20 until approximately the 210bp position.

5.1.3 Error rates

Overall error rates predicted by GemSIM, DRISEE, and BEAR when supplied with various types of WGS data are compared in Figure 5.5. GemSIM failed to report error rates for every base pair position in the Ion Torrent and 454 datasets, in particular predicting error rates of 0 for positions beyond 250 and 525, respectively. In order to generate errors for all positions in long reads, BEAR automatically performs a exponential regression on the predicted error rates. This frees the user from the need for parameter tuning.

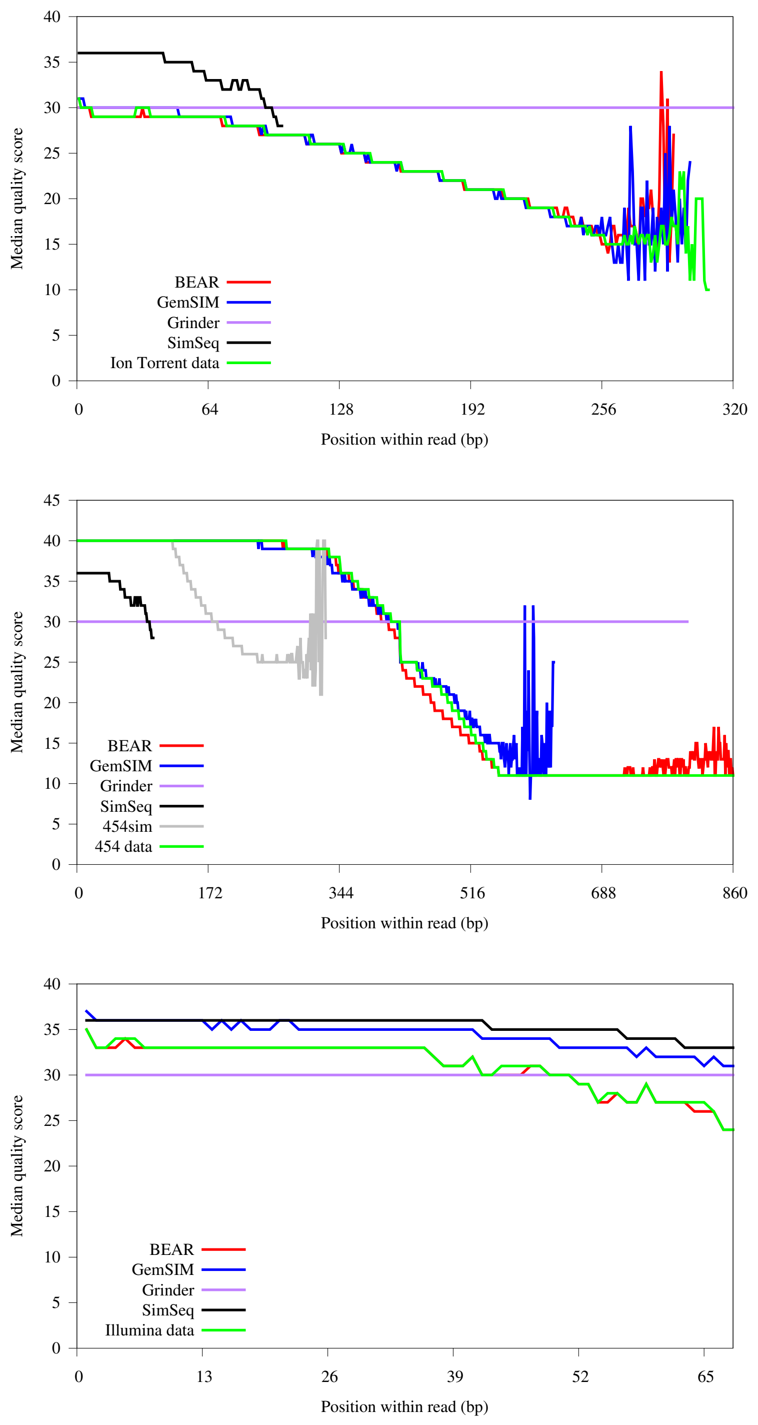


Figure 5.3: Comparison of quality score distributions for real and simulated WGS datasets. Top: Ion Torrent. Middle: 454. Bottom: Illumina. 454sim and MetaSim are excluded from the Ion Torrent and Illumina plots, as MetaSim does not generate quality scores and 454sim generated reads with a median quality score of 40 for all positions. GemSIM, 454sim, and SimSeq are unable to match the read length distribution of the 454 data, and as a result their quality score traces end prior to position 860.

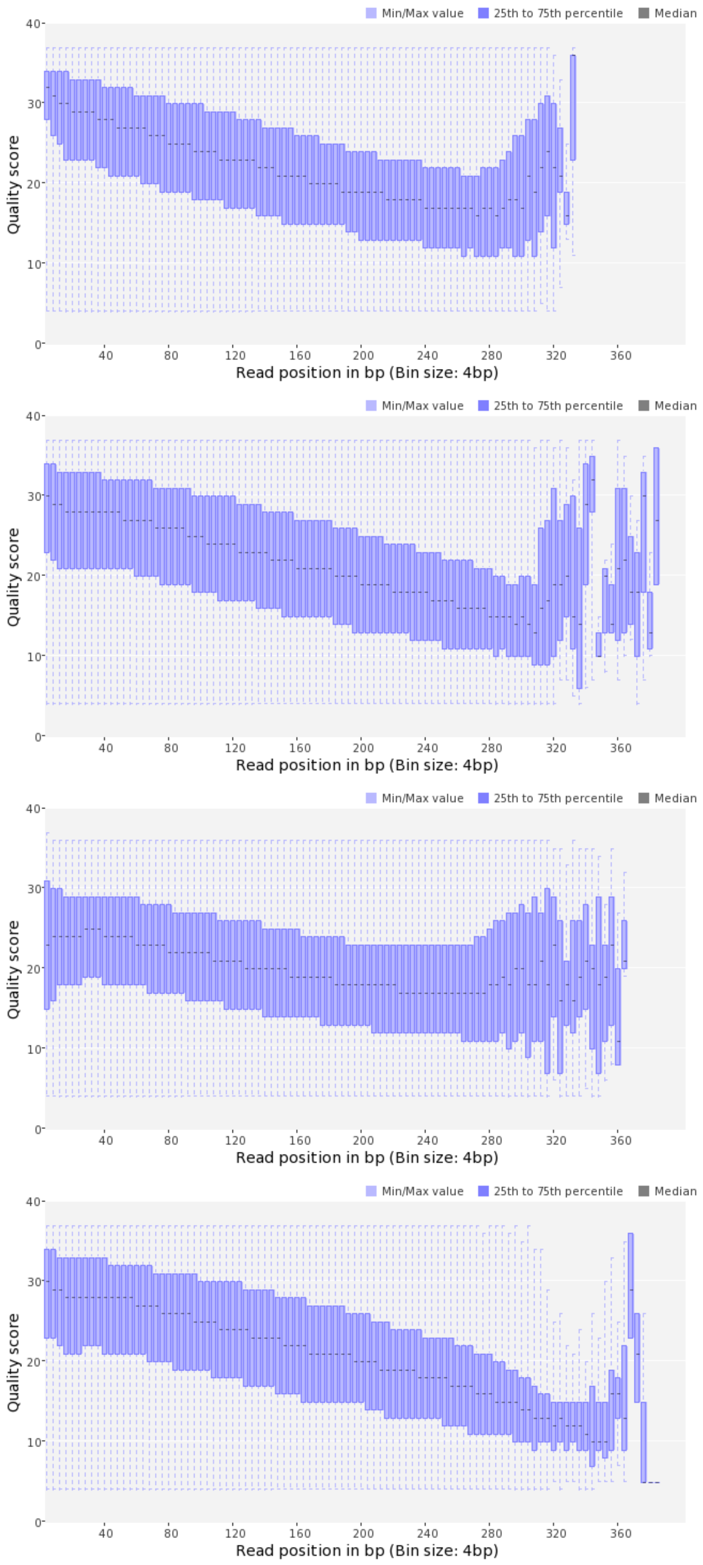


Figure 5.4: Comparison of quality score profiles generated by different types of Markov chains. Quality profiles are represented here in box plot format. From the top: First-order Markov chain using an average of previous qualities; First-order Markov chain; Second-order Markov chain; Real Ion Torrent metagenomic sequence data. Figures were generated using PRINSEQ [70].

This feature also allows BEAR to potentially generate substitution, insertion, and deletion errors at any possible read position, a feature that may not always be possible in GemSIM. GemSIM overestimated error rates at the beginning of all reads, and underestimated error rates at the ends of long reads (typically the most error prone region). GemSIM also overestimated the error rate at every position in the Illumina data. Conversely, BEAR predicted increases in error rates as read length increased for all datasets, with error models that more closely matched the real error rates.

5.1.4 Error-quality models

Error-quality models were generated for the datasets trained on the genomic Ion Torrent, 454, and Illumina datasets (each error-quality model consisting of four substitution error-quality models, and one indel error-quality model). Since BEAR is the only sequencing simulator that explicitly uses error-quality models, it is the only program featured in this section. Each model was evaluated by plotting the average quality scores from the real data for a given nucleotide and error type and superimposing the respective model on top of it. The R^2 values were used to determine how well each model fits to the respective dataset. The resulting models for the guanine nucleotide can be seen in Figure 5.6. The nucleotide-specific substitution errors did not appear to be substantially different for the Ion Torrent and 454/Pyrosequencing data, but there were significant differences in the nucleotide-specific substitution models for the Illumina data, as shown in Figure 5.7.

While there is sparse literature on the exact relationship between sequencing errors and quality scores for specific technologies, we did observe that our Ion Torrent error-quality model predicts that substitution errors would not have quality scores above Q20, which agrees with previously published results [10].

5.2 Effects of simple quality control measures on real and simulated reads

The effects of a simple quality control pipeline were measured on real and simulated Ion Torrent data, all consisting of 1,187,109 reads and approximately 200 million base pairs. Quality control statistics for each dataset are shown in Table 5.2. For each Ion Torrent dataset, all reads not within one standard deviation of the mean sequence length were removed. This step removed nearly 300,000 reads and over 30 million base pairs from the real dataset, with slightly less being removed from the GemSIM and BEAR datasets and far more being removed from the Grinder dataset. Less than 26 million base pairs were removed from the GemSIM and BEAR datasets, but nearly 65 million were removed from the Grinder dataset. The

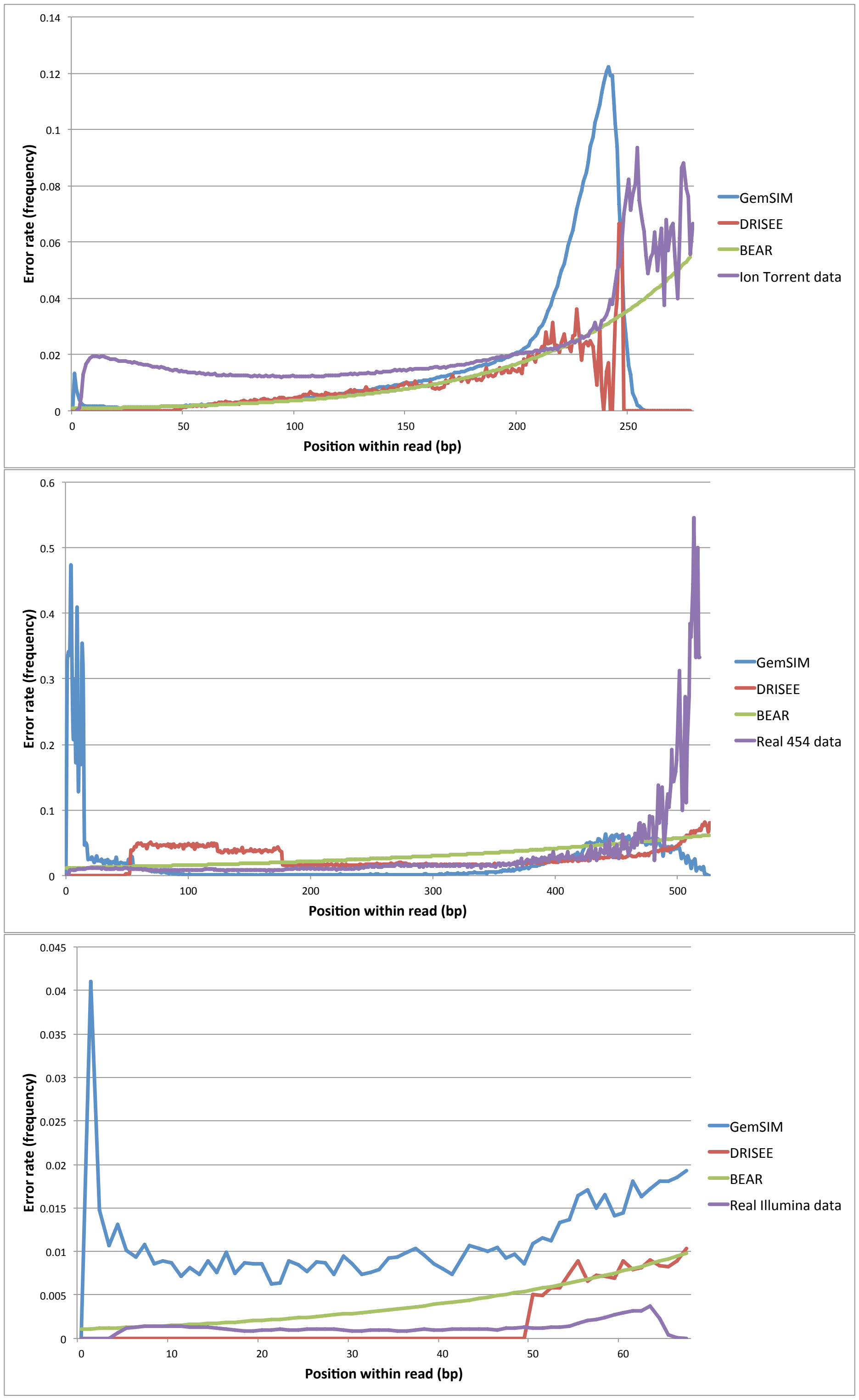


Figure 5.5: Overall error rates for real WGS data and error rates predicted by GemSIM, DRISEE, and BEAR. Top: Ion Torrent data; Middle: 454 data; Bottom: Illumina data. Real error rates were inferred by aligning reads to their respective reference genomes using Bowtie2. Error rates are displayed as relative frequencies.

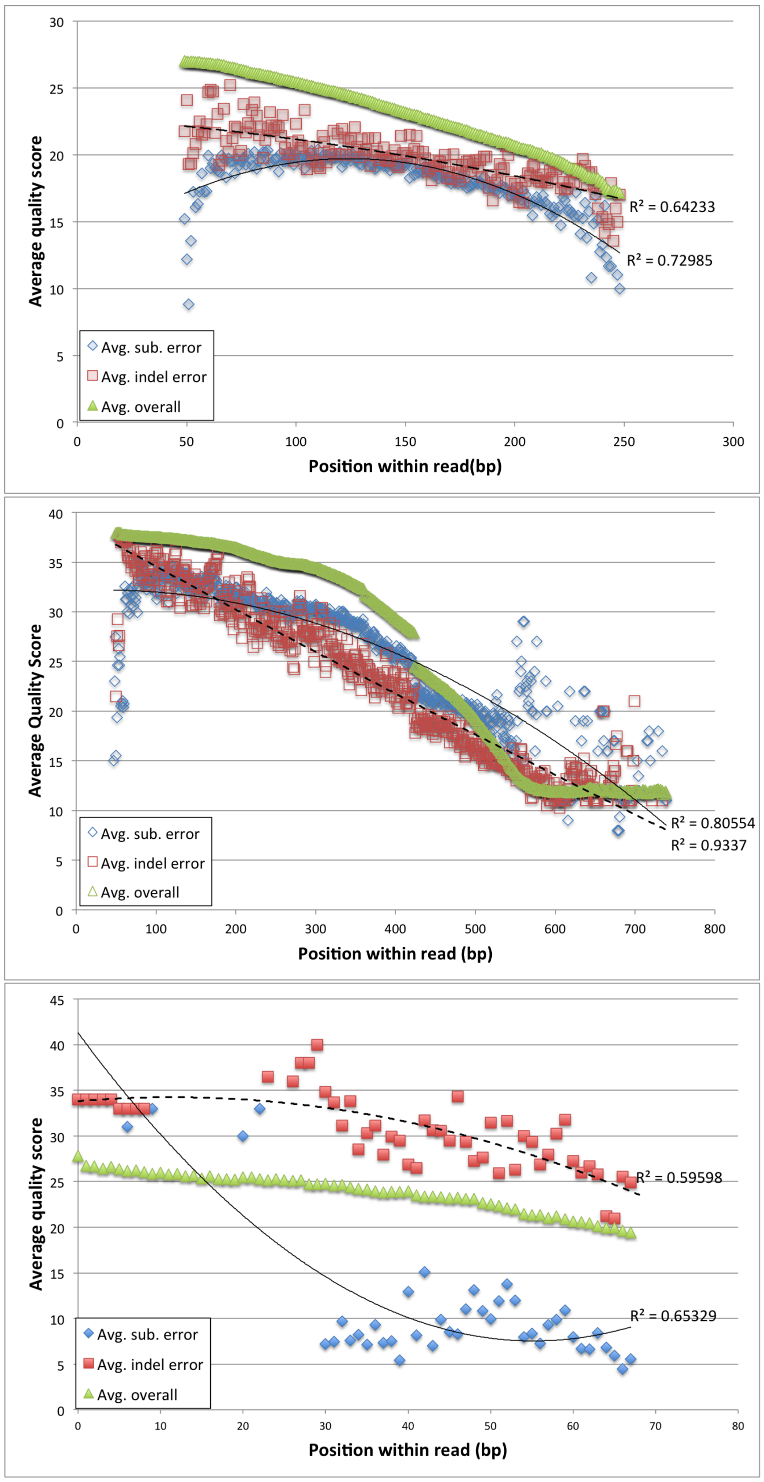
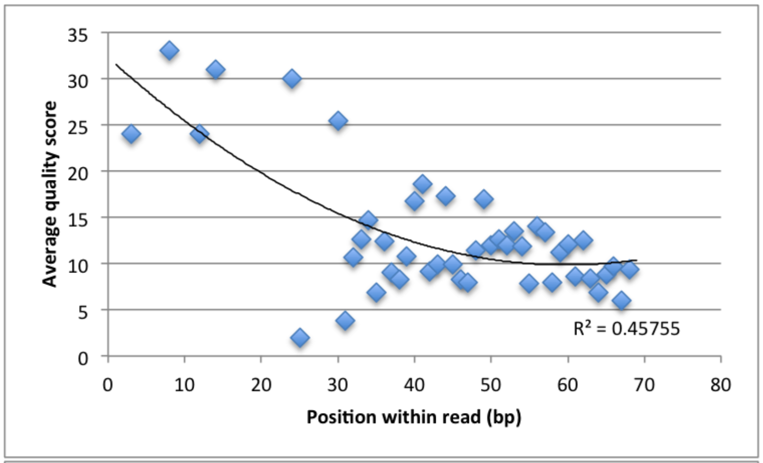
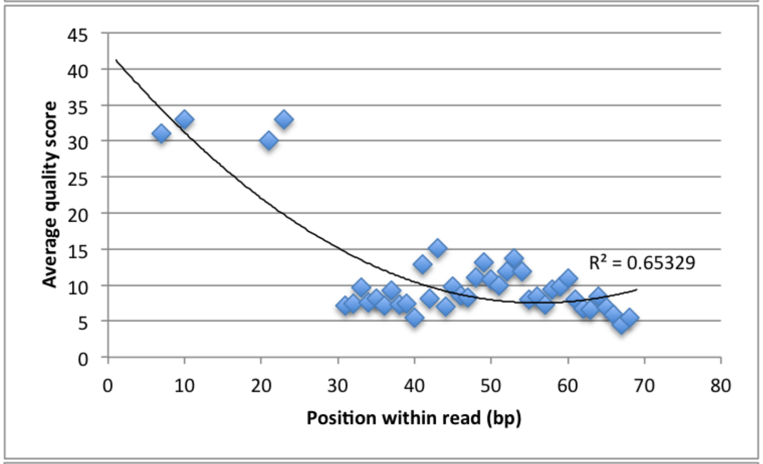


Figure 5.6: Substitution (solid line) and indel error-quality models (dashed line) generated by BEAR compared to average quality scores of erroneous nucleotides in real NGS data (red and blue points for substitution and indel errors, respectively). Average quality scores for all nucleotides are included for comparison (green points). Substitution error-quality models are generated for all 4 nucleotides by performing second-degree polynomial regression on error-quality data produced by BEAR. This figure shows the data from guanine substitution errors. Top: Ion Torrent; Centre: 454/Pyrosequencing; Bottom: Illumina.





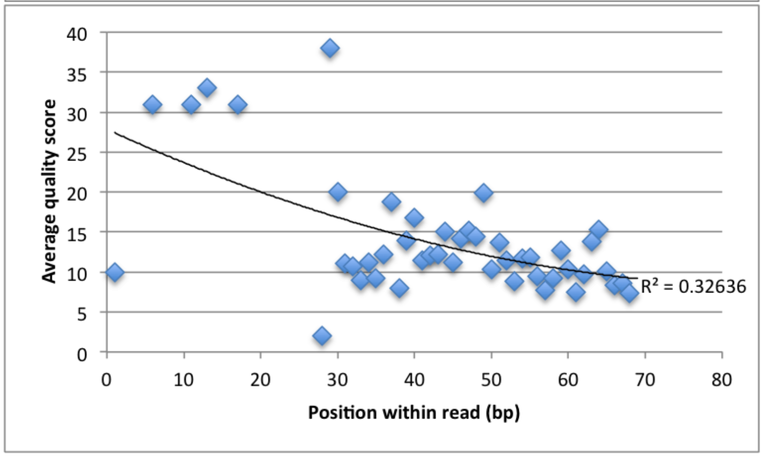


Figure 5.7: Substitution error quality models (black lines) for three nucleotides in Illumina data compared to actual quality scores (blue points). Top: Adenine; Centre: Cytosine; Bottom: Thymine. The model for guanine is part of Figure 5.6.

	Real data	**GemSim data**	**Grinder data**	**BEAR data**
# reads (# bp), pre-filtering	1,187,109 (199,694,957)	1,187,109 (205,286,714)	1,187,109 (199,609,471)	1,187,109 (201,422,315)
# reads (# bp) after length filtering	884,748 (169,063,370)	925,411 (180,214,916)	805,527 (134,783,929)	914,023 (175,584,857)
# reads (#bp) after quality filtering	884,748 (166,677,676)	925,411 (176,818,912)	805,527 (134,783,929)	914,023 (171,501,924)
% remaining reads (% bp)	74.53% (83.47%)	77.96% (86.13%)	67.86% (67.52%)	76.99% (85.15%)
Alignment rate	99.29%	100.00%	100.00%	100.00%
Average edit distance	4.60	3.62	0.61	2.02
Average % identity	97.36%	98.17%	99.63%	98.94%
Genome coverage	99.81%	99.94%	99.98%	99.97%

Table 5.2: Quality control statistics for real and simulated Ion Torrent data. Alignment rate refers to the total proportion of contigs that aligned to the reference genome. Average edit distance is the average number of insertions, deletions, and substitutions present in the contig when compared to the reference genome. The methodology for calculating average percent identity is described in Section 4.4.2. Genome coverage was calculated by aligning contigs to the reference genome and using the `mpileup` program found in SAMtools [46].

length-filtered datasets were then subjected to quality trimming, where each read would be trimmed to the first position where the average quality is below 20. This step did not remove any full reads from any of the datasets, but did remove between 2.4 and 4.1 million bp from the GemSIM, BEAR, and real datasets. No base pairs were removed from the Grinder dataset. All simulated datasets reported lower average edit distances and higher average percent identities, genome coverage, and alignment rates than the real dataset. In terms of the total amount of sequencing data removed as a result of quality control, the BEAR data most closely matched the real data. However, in terms of average percent identity and genome coverage, GemSIM most closely matched the real data.

5.3 Evaluation of assembly programs using real and simulated data.

The Velvet and MIRA assemblers were used to assemble simulated genomic Ion Torrent datasets from Grinder, GemSIM, and BEAR in addition to a real *E. coli* Ion Torrent dataset. Assembly statistics for Velvet and MIRA are found in Tables 5.3 and 5.4, respectively. Compared to MIRA, the Velvet assemblies had all lower N50 values, more contigs, and smaller assembly sizes. Furthermore, the Velvet assemblies all had higher average percent identities than those generated by MIRA, suggesting that Velvet results in a more accurate assembly. Additionally, the Velvet assembly of the BEAR data had an average percent identity closest to that of the real data, whereas the Velvet assembly of the GemSIM data had a more realistic genome coverage.

	Real data	**GemSim data**	**Grinder data**	**BEAR data**
# contigs	3,880	1,192	1,272	580
N50	1,507	6,581	6,621	15,356
Largest contig size	8,454	78,567	29,056	78,558
Assembly size	3,646,829	4,381,255	4,444,487	4,454,030
Avg. alignment rate	99.87%	99.92%	99.92%	99.83%
Avg. % identity	99.69%	99.88%	99.99%	99.79%
Genome coverage	81.54%	94.89%	95.45%	95.33%

Table 5.3: Results of Velvet assemblies for real and simulated genomic Ion Torrent data.

	Real data	**GemSim data**	**Grinder data**	**BEAR data**
# contigs	3,410	276	258	144
N50	6034	83,355	85,841	83,313
Largest contig size	37,657	236,668	326,619	352,736
Assembly size	5,457,632	4,563,018	4,554,200	4,508,725
Avg. alignment rate	98.77%	99.28%	100%	95.00%
Avg. % identity	99.56%	99.82%	99.91%	94.90%
Genome coverage	95.30%	94.64%	96.50%	77.85%

Table 5.4: Results of MIRA assemblies for real and simulated genomic Ion Torrent data.

By comparison, the MIRA assemblies were all larger than the respective Velvet assemblies. The Grinder dataset and real dataset had higher coverages when assembled by MIRA, whereas the BEAR and GemSIM had lower coverages. We also observed that the MIRA assemblies all had lower average percent identities than those generated by Velvet, particularly the BEAR dataset. Additionally, MIRA generated a strange warning message when assembling the BEAR dataset, shown in Figure 5.8. This warning messages suggests that MIRA had difficulty assembling the BEAR dataset, which is also supported by the relatively low alignment rate, average percent identity, and genome coverage for the BEAR dataset when compare to the other assemblies. Documentation for this error is sparse, but seems to indicate that it is caused by a currently unaddressed software bug in MIRA [15]. These results suggest that Velvet would be a more reliable general purpose assembler when compared to MIRA, particularly if the goal is to generate accurate contigs. If the goal of assembling is to generate a more complete assembly to the potential detriment of accuracy, MIRA may be the better assembler program. With respect to read simulators generating data that can produce realistic results from analysis, it would appear that BEAR and GemSIM both generate realistic results in certain criteria (average percent identity and genome coverage, respectively) when using Velvet, but results are inconclusive for MIRA as it was unable to assemble all of the datasets to a satisfactory degree.

5.3.1 Effects of GC profile emulation and sequencing errors on assembly

Three additional simulated datasets were assembled by Velvet to determine the effects of sequencing errors and GC profile emulation on assembly: one dataset resulting from biased sampling with no sequencing errors,

```
MIRA warncode: CONCOV_SUSPICIOUS_DISTRIBUTION
Title: Suspicious distribution of contig coverages


- 0 contig(s) with a total of 0 bases (= -nan\% of bases in all non-repetitive
  large contigs) have an average coverage less than 75\% of the average coverage
  of all non-repetitive large contigs.
- 0 contig(s) with a total of 0 bases (= -nan\% of bases in all non-repetitive
  contigs) have an average coverage more than 125\% of the average coverage of
  all non-repetitive large contigs.
- 0 contig(s) with a total of 0 bases (= -nan\% of bases in all non-repetitive
  contigs) have an average coverage 25\% above or below the average coverage of
  all non-repetitive large contigs.
Summary: found 3 indicator(s) for coverage problem(s).


If the DNA you are assembling is bacterial, this could indicate that you sampled
and sequenced DNA from exponential or late exponential phase of a bacterial
population. This leads to a coverage bias toward the origin of replication,
hence false positive detection of repeats, hence an assembly which is more
fragmented than it could be or may have misassemblies in regions located toward
the opposite of the origin of replication.
Only available countermeasure: for your next sequencing project, do not sample
in exponential phase but sample in stationary phase (if possible).
```

Figure 5.8: Critical warning message produced by MIRA when assembling simulated genomic Ion Torrent data generated by BEAR.

	Real data	**Bias, no errors**	**No bias, no errors**	**No bias, errors**	**Bias, errors**
# contigs	3,880	311	219	580	1,891
N50	1507	40,263	78,894	15,356	3,935
Largest contig size	8,454	225,820	326,338	78,558	16,748
Assembly size	3,646,829	4,393,893	4,483,477	4,454,030	3,919,070
Avg. alignment rate	99.87%	99.68%	98.63%	99.83%	100.00%
Avg. % identity	99.69%	99.68%	99.92%	99.79%	99.71%
Genome coverage	81.54%	94.05%	92.87%	95.33%	85.54%

Table 5.5: Results of Velvet assemblies for biased real data, and both biased and unbiased simulated data.

one with both biased sampling and sequencing errors, and one without biased sampling or sequencing errors. The assembly statistics for these three datasets compared to the real data and an unbiased-with-errors dataset (labelled 'BEAR data' in Table 5.3) can be found in Table 5.5. From these results, we can see that generating data with either sequencing errors or biased sampling can affect assembly by reducing the accuracy of the contigs, increasing the total number of contigs, and decreasing the maximum contig size and N50 value. However, it appears that these effects are compounded when using a dataset with both sequencing errors and biased sampling. Additionally, including both biased sampling and sequencing errors in a simulated dataset can substantially reduce the total genome coverage of the contigs and the total assembly size. Furthermore, we observe that, when considering all seven assembly statistics as a whole, the simulated dataset with both biased sampling and sequencing errors more closely matches the assembly of the real data than any assembly of simulated data in Tables 5.3 or 5.5. This suggests that GC profile emulation and sequencing errors both may be necessary components of a given sequencing data simulation strategy. However, of all the programs evaluated in this thesis, only BEAR is able to emulate both error rates and GC profile emulation.

5.4 Evaluation of metagenomics classification tools with BEAR

Three simulated metagenomics datasets generated by BEAR were used to compare the classification tools MetaPhlAn and MEGAN. The first metagenomic dataset contained reads sampled from the 2,062 bacterial genomes present in the NCBI Genomes database following a low-evenness abundance profile. The results of the classification tools can be found in Figure 5.9. From this dataset, MetaPhlAn classified the reads to only four species while MEGAN failed to classify any reads at the species level.

The second dataset had the same characteristics as the first, but each individual genome was subjected to biased sampling based on the GC profile of real Ion Torrent metagenomic data made available to our lab. The results from classifying these reads are shown in Figure 5.10. For this dataset, both tools classified the reads with an abundance profiles that would be typical of low-evenness environments with MetaPhlAn predicting a range of abundances on the order of 4 logs. However, MetaPhlAn greatly overestimated the degree to which the most abundant species was present, with over two-thirds of the reads being classified to a single species. Additionally, MetaPhlAn only predicted that 27 species were present in the sample. MEGAN performed better on this dataset, predicting 1,879 species in the dataset. MEGAN exhibited the opposite behaviour of MetaPhlAn by underestimating the dominance of the two most abundant species in the dataset.

The third dataset consisted of 122 genomes present in equal abundances. This dataset used the same GC profile as the previous one, but with the sampling applied across all genomes at once so that the abundances of each genome could be affected by the biased sampling. Both MEGAN and MetaPhlAn overestimated the number of species in the dataset, classifying the reads to 1,036 and 277 species, respectively. MetaPhlAn overestimated the abundances of the four most prominent organisms to a greater degree than MEGAN, but MEGAN overestimated the abundances of the 45 subsequent organisms that MetaPhlAn had accurately predicted.

Overall, both programs performed far better on the datasets that had incorporated biased sampling during simulation. The dataset with unbiased sampling was classified very poorly by both programs (Figure 5.9). This would be contrary to expectations, as it would be expected that an unbiased dataset would be 'simpler' and therefore easier to classify. Of the two datasets that used biased sampling, MEGAN appeared to predict a more accurate abundance profile for the second dataset (Figure 5.10) while MetaPhlAn performed better on the third dataset (Figure 5.11). Thus, it appears that the results of this analysis are inconclusive, as neither of the two programs clearly outperformed the other. However, Figure 5.11 does provide support for our methodology for performing biased sampling, as neither program predicted that all organisms are present with equal abundances.

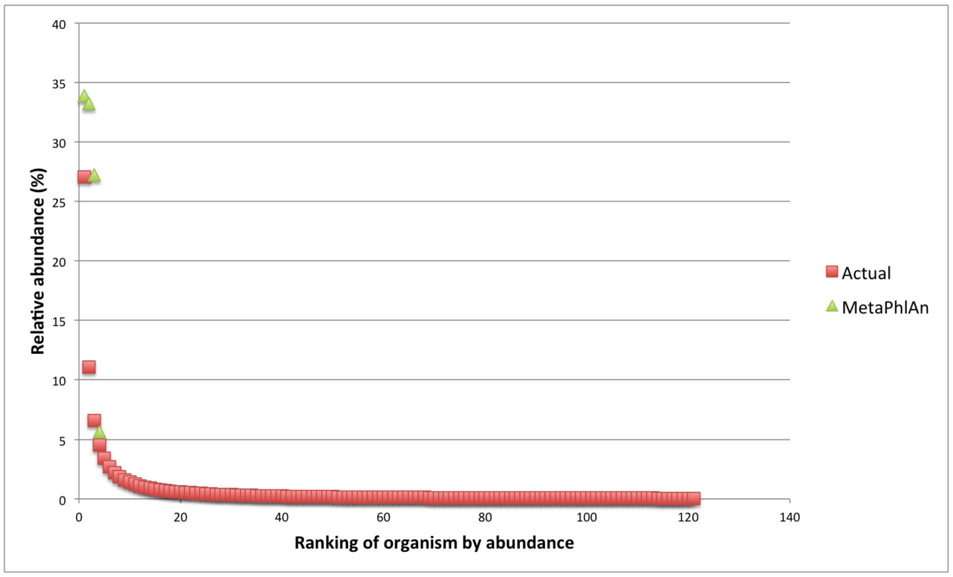


Figure 5.9: Results of classification for a simulated metagenomic dataset consisting of 1 million reads from 2,062 genomes. This figure only shows results up to the first 122 genomes. Organisms were sorted by their relative abundance and assigned a ranking based on this abundance.

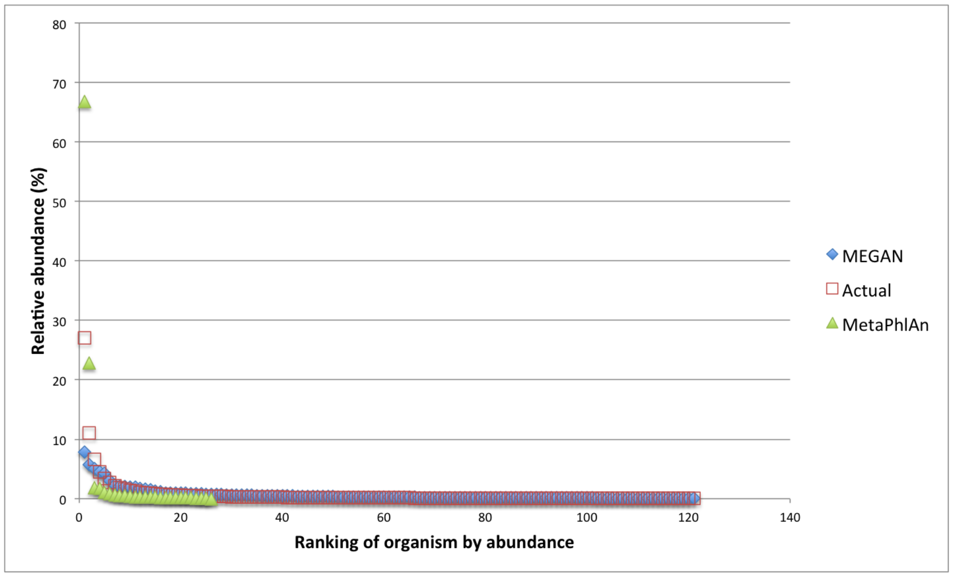


Figure 5.10: Results of classification for a simulated metagenomic dataset consisting of 1 million reads from 2,062 genomes each being individually subjected to biased sampling. This figure only shows results up to the first 122 genomes. Organisms were sorted by their relative abundance and assigned a ranking based on this abundance.

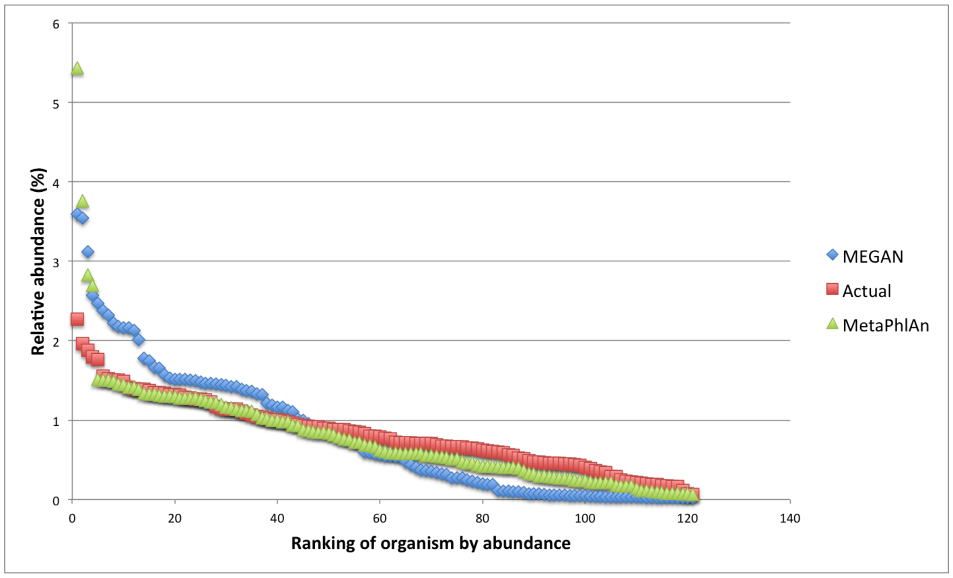


Figure 5.11: Results of classification for a simulated metagenomic dataset consisting of 1 million reads from 122 concatenated genomes subjected to biased sampling. This figure only shows results up to the first 122 genomes identified by the classifiers. MEGAN and MetaPhlAn classified the reads to 1,036 and 277 species, respectively. Organisms were sorted by their relative abundance and assigned a ranking based on this abundance.

Chapter 6

Discussion, Conclusion, and Future Work

In this thesis, we presented a software suite called BEAR which provides improved emulation of (meta)genomic sequence reads. BEAR was compared to other sequence-read simulators, and various analyses were performed on real and simulated data to evaluate the programs' simulation strategy. In Section 6.1, we discuss and interpret the results presented in Chapter 5. In Section 6.2, we summarize the conclusions that can be drawn from the results. The chapter ends with Section 6.3, in which we present a number of potential avenues and possibilities for future work with BEAR.

6.1 Discussion

6.1.1 BEAR provides improved emulation of reads without aligning to a reference genome

The results in the previous chapter suggest that BEAR can be particularly useful for simulating raw genomic reads from NGS technologies such as Ion Torrent, which exhibit characteristics that most current programs are unable to emulate well. The results presented in Table 5.2 demonstrate the importance of properly emulating the read length distribution of a set of reads. While GemSIM and BEAR almost perfectly modelled the distribution (see Figure 5.2), Grinder could only generate reads following a normal distribution. By subjecting the real and simulated reads to quality control measures, we were able to observe that filtering by read length removed far too many reads from the Grinder dataset (nearly one-third of the reads removed) while the GemSIM and BEAR datasets were quite close to the real data, in the range of 23% to 25.5% of the total reads being removed. Additionally, this same table demonstrates the importance of emulating quality scores correctly when subjecting simulated data to quality score-based filtering. The GemSIM, BEAR, and real datasets all had between 2.4 and 4 million base pairs removed by quality filtering, but the simple quality score model used by Grinder failed to result in even a single nucleotide being removed by the quality filter.

6.1.2 Cases where BEAR outperforms GemSIM

Figure 5.3 suggests that the general decline in median quality across the length of the read in actual Ion Torrent data is captured both by our position-dependent Markov chain-based approach, and the alignment-based context-dependent method used by GemSIM. However, for the 454 and Illumina datasets where the reads did not align as well to the reference genome, BEAR clearly emulated the read length and quality distributions better than GemSIM. This may be because generating realistic reads in GemSIM is largely dependent on a set of reads aligning accurately to a reference genome, and in the case of the 454 and Illumina data, the reads were not as high-quality as the Ion Torrent reads. The dependence on a "good alignment" allows us to propose two situations where BEAR could be used instead of GemSIM:

1. BEAR may be quite useful for emulating "questionable" sequence runs, in which the overall quality of the reads may not be simple to ascertain. These low-quality reads are likely to align poorly (or not at all) to a reference genome. Reads with poor (or no) alignment would not be included in the construction of the GemSIM error model, excluding a large portion of reads and resulting in an unrealistic simulation. BEAR may be more appropriate in this situation, as it does not rely on alignment to a reference genome to construct its error model.

2. Direct emulation of characteristics of metagenomic sequencing runs. If one were to create an error model using metagenomic data in GemSIM, it would be necessary to align the metagenomic reads to all genomes in a database. This can be an extremely time-consuming operation if the number of reads and number of genomes are sufficiently large. Additionally, if there are reads from novel organisms in the sequence run which are not closely related to the organisms in the database, then there is a good chance that they will not align properly to any genomes. This, much like the previous item, can lead GemSIM to create an incorrect error model. Again, this problem may be avoided by BEAR provided there is a sufficient number of duplicate reads in the dataset.

6.1.3 BEAR and GemSIM: a new family of sequencing simulators

BEAR (like GemSIM) is capable to adapting to changes in NGS technology. For example, if the technology for a sequencing platform is modified to extend read length and quality characteristics, BEAR would be able to generate simulated data with these new qualities with no modifications. We also demonstrated that BEAR performs well with both genomic and metagenomic data, exhibiting versatility that is lacking in other existing programs. Thus, we believe that BEAR belongs in a new category of sequencing simulator programs without the need for external parameter calibration.

6.1.4 State-of-the-art sequence-read simulators still underperform compared to real data

In Sections 5.2 and 5.3, we attempted to compare the results of quality control pipelines and sequence assemblers using real and simulated data. While GemSIM and BEAR were able to achieve realistic results from the quality control analysis, nearly all assemblies of simulated data were ‘better’ in all metrics than the assemblies of the real data. This is a commonly observed phenomenon, and it is often argued that simulated sequence data is too clean, and results from an analysis using simulated data are unrealistically complete and accurate. Although subsequent generations of sequencing simulator programs use more robust, sophisticated error models and use techniques that directly emulate other characteristics of a sample of reads (e.g., GemSIM and BEAR), we still observe that even these new sequencing simulators can fail to generate results closely resembling those of real data. We propose two reasons for why this may be:

1. It may be the case that simulated data allows for complete results to be derived from incomplete, real data. Using genome assembly as an example, there are many regions of a genome that are extremely difficult to assemble due to being composed entirely of long, repetitive, low-complexity strings of nucleotides. These repeat regions are extremely problematic for assembler programs and are often ignored, as read lengths are often too short to extend beyond the repetitive region. This results in many published genome assemblies having long stretches of nucleotides that are not fully sequenced (usually denoted as an ’N’ placeholder nucleotide in FASTA format, if at all). Given that these low complexity regions are not present in the genomes that sequence-read simulator programs sample reads from, it would then be expected that a given analysis derived from this simulated data would be easier for a program to perform, as there is only high-complexity, informative sequences from which to derive results.

2. We observed that the best (i.e., most realistic) assembly of simulated data came from the BEAR reads that incorporated both sequencing errors and a GC profile. Thus, it is likely the case that it is not enough for a given sequence-read simulator program to simply randomly sample reads from genomes and superimpose an error model on top of the read; sequencing biases must be taken into account as well.

Furthermore, while surveying the literature that introduced the sequencing simulators used in this paper, we observed that none of the papers included any sort of analysis in which the results using simulated data were compared to real data. Rather, the level of analysis provided was similar to the results presented Section 5.1 of this thesis, where the characteristics of the raw simulated data were compared to the characteristics of the real data. As observed in Section 5.2 and 5.3, this level of analysis is an insufficient demonstration of the efficacy of a simulation strategy used by a given simulator program. Although simulated sequence-read data generated by different programs can produce different results, the results of analyses using simulated data from the best simulators tended to resemble those using data from 'worse' simulators rather than the results one gets from using real data.

Another larger issue with sequencing simulation is that, while it is well-established the volume of biological sequence data is constantly increasing, the same can not currently be said for the quality of data. Although manufacturers claim lower error rates with subsequent iterations of each sequencing technologies, core problems (e.g., homopolymer errors for 454 and Ion Torrent sequencing, GC bias, sequencing artifacts, low-complexity region sequencing) still remain and lack in-depth characterization. Even with improved error rates, pre-sequencing/preparatory processes such as PCR have been known to be a significant cause of sequencing errors, particularly for amplicon and single-genome studies where amplification is a necessary process to obtain sufficient sequencing depth. That is, it could also be argued that real biological sequence data is still too noisy, and that results from analyses using this data failing to account for this "noise" risk being incomplete.

6.2 Conclusion

This paper presented BEAR, a tool for generating simulated reads based on empirically-derived read length distributions and quality scores. The approach used by BEAR for generating data eliminates the need for parameter tuning, allowing for an easy-to-use interface; at minimum, the user need only provide a sample of data that has the desired properties of the reads to be emulated, the number of reads to generate, and the genome(s) from which to sample reads. We demonstrated that BEAR is superior to popular, existing artificial read generation programs in terms of producing reads with realistic read length and quality score distributions. While state-of-the-art programs such as GemSIM give comparable results in this regard, BEAR has additional features that make it more suitable for metagenomics applications, such as automatically producing community profiles and the lack of reliance on a reference genome. This also makes BEAR suitable for emulating any type of WGS sequencing run where the reads do not align sufficiently well to a reference genome. We have demonstrated that BEAR can be used for metagenomics research, and we believe that one of the best uses for BEAR will be for simulating metagenomic reads from emerging and consistently-updated technologies such as Ion Torrent, as there are few programs available that can capture their behaviour with respect to read length and overall quality scores. Furthermore, the tools included with BEAR are not just limited to metagenomics, as we have demonstrated that it may be useful for evaluating assembler programs, which are usually used for genomic data. Additionally, we present an alignment-free algorithm that allows for the GC profile of a given set of reads to be emulated, which can prove useful for evaluating the effects that GC bias may have on the results of a given analysis.

6.3 Future work

In this section, we the following potential avenues and improvements that could be made to BEAR:

- BEAR could be expanded to directly emulate amplicon datasets. As mentioned in Section 2.6, amplicon sequencing is commonly used to estimate the number of unique species in a given sample by sequencing only the well-conserved genetic sequences unique to each organism (e.g., 16S rRNA gene has regions that bind to ‘universal primers’ and have regions unique to each species) and then amplifying the sequences. Grinder is currently the only sequencing simulator program that can simulate amplicon sequencing, so identifying shortcomings in the Grinder approach (e.g., simplified error models, lack of amplicon sequencing-specific biases) could be used to inform the design of this feature in BEAR.

- Previous work by Ross et al. demonstrated that sequences with extreme GC content values had higher error rates than sequences with more moderate GC content values [65]. BEAR, then, could have two error models: one for high/low GC content sequences with higher error rates, and an error model with more moderate error rates for the sequences with more moderate GC contents.

- In this thesis, we only compared the simulated reads to real Ion Torrent, pyrosequencing, and Illumina reads. As mentioned in Section 2.2.4, there are a number of emerging technologies that may soon be commonplace. Thus, it may prove useful to evaluate the ability of these programs to simulate, for example, reads generated by SMRT sequencing. The extremely long read lengths may provide new challenges for this domain, and current approaches used by sequencing simulators (e.g., the alignment-based approach used by GemSIM) may be too computationally demanding for this task.

- Current sequencing simulators operate under the assumption that errors are distributed uniformly among all reads. That is, some arbitrary read r_1 in a sample S would have the same error model as another arbitrary read r_2. However, previous work by Huse et al. demonstrated that this is not necessarily the case [32]. The authors observed of their dataset: “82% [of the reads] had no errors, 93% had no more than a single error, and 96% had no more than 2 errors. Conversely, a small number of reads, fewer than 2%, contained a disproportionate number of errors that account for nearly 50% of the miscalls for the entire dataset." That is to say, a small proportion of the reads have most of the sequencing errors while a majority of the reads have little or no errors. It may prove useful to develop a method that could identify this small subset of error-prone reads in a sample (preferably without aligning to a reference genome, for metagenomics experiments) so that BEAR could have separate error models for these error-prone and not-error-prone reads. If the proportion of "bad" reads in a dataset could be inferred, the simulated reads could be classified as "good" or "bad" and then have the appropriate error model superimposed on each read.

- Similarly to the second point, it may be worth investigating to see if certain "classes" of quality strings exist. In its current form, BEAR uses the same Markov chain to generate quality strings for all reads in a sample. However it may be the case, for example, one class of reads exhibits a simple linear decrease in quality while a second class exhibits a sharp drop-off in quality after a certain position, and a third class of reads may just be "random noise" (i.e., no clear trend between the quality score and position within the read). If this could be experimentally validated, BEAR could then have one Markov chain per "class" of reads.

- Following up on the hypothesis proposed in Section 6.12, it may prove interesting to perform some sort of complexity analysis to address the fact that simulated reads may be lacking in low-complexity sequences due to the difficulty of assembling low-complexity sequence regions. This would require the development of robust comparative metrics for DNA sequence complexity such that the complexity of a published genome sequence (or simulated reads derived from the genome) could be compared to the complexity of a set of unassembled real reads. If our analysis demonstrates that there is, in fact, a difference between the complexity of real reads and the complexity of the genome, then would be useful to determine what proportion of the real reads are composed of these low complexity regions and develop methods that could increase the presence of low-complexity regions into the simulated data.

- BEAR currently uses a command-line interface, which can be cumbersome for many users. In the future it would be useful to design a GUI to execute the programs used by BEAR. Additionally, it would likely be convenient for many users if the GUI was implemented using the widely-used Galaxy workflow environment [27].

References

[1] D Aird, MG Ross, WS Chen, M Danielsson, T Fennell, C Russ, DB Jaffe, C Nusbaum, and A Gnirke. Analyzing and minimizing PCR amplification bias in Illumina sequencing libraries. *Genome Biology*, 12:R18, 2011.

[2] FW Allen. The Biochemistry of the Nucleic Acids, Purines, and Pyrimidines. *Annual Review of Biochemistry*, 10:221–1146, 1941.

[3] FE Angly, D Willner, F Rohwer, P Hugenholtz, and GW Tyson. Grinder: a versatile amplicon and shotgun sequence simulator. *Nucl. Acids Res.*, 40(12):e94, 2012.

[4] FE Angly, D Willner, F Rohwer, P Hugenholtz, and GW Tyson. `http://sourceforge.net/projects/biogrinder/files/biogrinder/Grinder-0.4.7/`, 2012. [Online, accessed May 25, 2014].

[5] S Balzer, K Malde, and I Jonassen. Systematic exploration of error sources in pyrosequencing flowgram data. *Bioinformatics*, 27(13):i304–i309, 2011.

[6] S Balzer, K Malde, A Sharma, and I Jonassen. Characteristics of 454 pyrosequencing data–enabling realistic simulation with flowsim. *Bioinformatics*, 26(18):i420–5, 2010.

[7] JMS Bartlett and D Stirling. A Short History of the Polymerase Chain Reaction. *PCR Protocols*, 226:3–6, 2003.

[8] DR Bentley, S Balasubramanian, HP Swerdlow, GP Smith, and J Milton et al. Accurate Whole Genome Sequencing using Reversible Terminator Chemistry. *Nature*, 456(7218):53–59, 2008.

[9] KR Bradnam, JN Fass, A Alexandrov, P Baranay, M Bechner, I Birol, S Boisvert, JA Chapman, G Chapuis, R Chikhi, H Chitsaz, WC Chou, J Corbeil, C Del Fabbro, TR Docking, R Durbin, D Earl, S Emrich, P Fedotov, NA Fonseca, G Ganapathy, RA Gibbs, S Gnerre, E Godzaridis, S Goldstein, M Haimel, G Hall, D Haussler, JB Hiatt, and IY Ho et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2:10, 2013.

[10] LM Bragg, G Stone, MK Butler, P Hugenholtz, and GW Tyson. Shining a Light on Dark Sequencing: Characterizing Errors in Ion Torrent PGM data. *PLoS Comp. Biol.*, 9(4):e1003031, 2013.

[11] J Brodin, M Mild, C Hedskog, E Sherwood, T Leitner, B Andersson, and J Albert. PCR-Induced Transitions Are the Major Source of Error in Cleaned Ultra-Deep Pyrosequencing Data. *PLoS One*, 8(7):e70388, 2013.

[12] MJ Chaisson and PA Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res.*, 18:324–330, 2007.

[13] YC Chen, T Liu, CH Yu, TY Chaing, and CC Hwang. Effects of GC Bias in Next-Generation-Sequencing Data on *De Novo* Genome Assembly. *PLoS One*, 8(4):e62856, 2013.

[14] B Chevreux. MIRA: An Automated Genome and EST Assembler. *German Cancer Research Center, Heidelberg*, 2005.

[15] B Chevreux. `http://sourceforge.net/p/mira-assembler/tickets/7/`, 2014. [Online, accessed July 7, 2014].

[16] FH Crick. Central Dogma of Molecular Biology. *Nature*, 227:561–563, 1970.

[17] J Dabney and M Meyer. Length and GC-biases during sequencing library amplification: A comparison of various polymerase-buffer systems with ancient and modern DNA sequencing libraries. *BioTechniques*, 52(2):87–94, 2012.

[18] R Dahm. Discovering DNA: Friedrich Miescher and the early years of nucleic acid research. *Human Genetics*, 122(6):565–581, 2008.

[19] JC Dohm, C Lottaz, T Borodina, and H Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucl. Acids Res.*, 36(16):e105, 2008.

[20] D Earl, K Bradnam, J St John, A Darling, D Lin, J Fass, HO Yu, V Buffalo, DR Zerbino, M Diekhans, N Nguyen, PN Ariyaratne, WK Sung, Z Ning, M Haimel, JT Simpson, NA Fonseca, I Birol, TR Docking, IY Ho, DS Rokhsar, R Chikhi, D Lavenier, G Chapuis, D Naquin, N Maillet, MC Schatz, DR Kelley, AM Phillippy, and S Koren et al. Assemblathon 1: A competitive assessment of short read assembly methods. *Genome Res.*, 21:2224–2241, 2011.

[21] RC Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, 2010.

[22] Editorial. The human genome at ten. *Nature*, 464:649–650, 2010.

[23] J Eid, A Fehr, J Gray, K Luong, J Lyle, G Otto, P Peluso, and D Rank et al. Real-Time DNA Sequencing from Single Polymerase Molecules. *Science*, 323(5910):133–138, 2008.

[24] B Ewing, L Hillier, MC Wendl, and P Green. Base-calling of automated sequencer traces using phred. I. Accuracy assessment. *Genome Res.*, 8(3):175–185, 1998.

[25] W Fiers, R Contreras, F Duerinck, G Haegeman, D Iserentant, and J Merregaert et al. Complete nucleotide sequence of bacteriophage MS2 RNA: primary and secondary structure of the replicase gene. *Nature*, 260(5551):500–507, 1976.

[26] C Gabriel, D Furst, I Fae, S Wenda, C Zollikofer, J Mytilineos, and GF Fischer. Hla typing by next-generation sequencing getting closer to reality. *Tissue Antigens*, 83(2):65–75, 2014.

[27] J Goecks, A Nekrutenko, J Taylor, and T Galaxy Team. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[28] V Gomez-Alverez, TK Teal, and TM Schmidt. Systematic artifacts in metagenomes from complex microbial communities. *The ISME Journal*, 3:1314–1317, 2009.

[29] MA Grohme, RF Soler, M Wink, and M Frohme. Microsatellite marker discovery using single molecule real-time circular consensus sequencing on the Pacific Biosciences RS. *BioTechniques*, 55(5):doi:10.2144/000114104, 2013.

[30] Y Guo, C Li, J Long, DC Samuels, and Y Shyr. The effect of strand bias in illumina short-read sequencing data. *BMC Genomics*, 13:666, 2012.

[31] X Huang and A Madan. CAP3: A DNA Sequence Assembly Program. *Genome Res.*, 9(9):868–877, 1999.

[32] SM Huse, JA Huber, HG Morrison, ML Sogin, and DM Welch. Accuracy and quality of massively parallel DNA pyrosequencing. *Genome Biol*, 8:R143, 2007.

[33] DH Huson, AF Auch, J Qi, and SC Schuster. MEGAN Analysis of Metagenomic Data. *Genome Research*, 17:377–386, 2007.

[34] L Janin, G Rosone, and AJ Cox. Adaptive reference-free compression of sequence quality scores. *Bioinformatics*, 30(1):24–30, 2014.

[35] LJ Jensen, C Friis, and DW Ussery. Three views of microbial genomes. *Research in Microbiology*, 150(9-10):773–777, 1999.

[36] S Johnson. Analysis and Comparison of Filtering Techniques for Metagenomic Data. Unpublished technical report, Department of Computer Science, University of Saskatchewan.

[37] S Johnson, B Trost, and A Kusalik. Improved quality score generation for erroneous nucleotides in simulated (meta)genomic data, 2014. Poster session presented at HiTSeq 2014; 2014 July 11-12; Boston, MA.

[38] S Johnson, B Trost, JR Long, and A Kusalik. A better sequence-read generator program for metagenomics. *journal.embnet.org*, 19.A:49, 2013. Poster abstract from The Next NGS Challenge Conference; 2013 May 14-16; Valencia, Spain.

[39] S Johnson, B Trost, JR Long, V Pittet, and A Kusalik. A better sequence-read simulator program for metagenomics. *Proceedings from RECOMB-Seq: Fourth annual RECOMB satellite workshop on massively parallel sequencing*, pages 246–257, 2014.

[40] JJ Kasianowicz, E Brandin, D Branton, and DW Deamer. Characterization of individual polynucleotide molecules using a membrane channel. *Proc Natl Acad Sci U S A*, 93(24):13770–13773, 1996.

[41] KP Keegan, WL Trimble, J Wilkening, A Wilke, T Harrison, M D'souza, and F Meyer. A platform-independent method for detecting errors in metagenomic sequencing data: DRISEE. *PLoS Comp. Biol.*, 8:e1002541, 2012.

[42] E Kellenberger. Bacterial Chromosome. *eLS*, 2006. doi:10.1038/npg.els.0004342.

[43] B Langmead and S Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9:357–359, 2012.

[44] J Lederberg. 'Ome Sweet 'Omics - A Genealogical Treasury of Words. *The Scientist*, 15:8, 2001.

[45] H Li, B Handsaker, A Wysoker, T Fennell, J Ruan, N Homer, G Marth, G Abecasis, and R Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[46] H Li, B Handsaker, A Wysoker, T Fennell, J Ruan, N Homer, G Marth, G Abecasis, and R Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[47] Z Li, Y Chen, and D Mu. Comparison of the two major classes of assembly algorithms: overlap-layout consensus and de-bruijn-graph. *Brief Funct Genomics*, 11(1):25–37, 2011.

[48] J Lightfield, NR Fram, and B Ely. Across Bacterial Phyla, Distantly-Related Genomes with Similar Genomic GC Content Have Similar Patterns of Amino Acid Usage. *PLoS One*, 6(3):e17677, 2011.

[49] MS Lindner and BY Renard. Metagenomic abundance estimation and diagnostic testing on species level. *Nucleic Acids Research*, 41(1):e10, 2013.

[50] L Liu, Y Li, S Li, Y He, R Pong, D Lin, L Lu, and M Law. Comparison of Next-Generation Sequencing Systems. *Journal of Biomedicine and Biotechnology*, 2012:1–11, 2012.

[51] NJ Loman, RV Misra, TJ Dallman, C Constantinidou, SE Gharbia, J Wain, and MJ Pallen. Performance comparison of benchtop high-throughput sequencing platforms. *Nature Biotechnology*, 30:434–439, 2011.

[52] C Luo, D Tsementzi, N Kyrpides, T Read, and KT Konstantinidis. Direct Comparisons of Illumina vs. Roche 454 Sequencing Technologies on the Same Microbial Community DNA Sample. *PLoS One*, 7(3):10.1371/journal.pone.0030087, 2012.

[53] F Lysholm, B Andersson, and B Persson. An efficient simulator of 454 data using configurable statistical models. *BMC Research Notes*, 4:449, 2011.

[54] K Mavromatis, N Ivanova, K Barry, H Shapiro, E Goltsman, EC McHardy, I Rigoutsos, A Salamov, F Korzeniewski, M Land, A Lapidus, I Grigoriev, P Richardson, P Hugenholtz, and NC Kyrpides. Use of simulated data sets to evaluate the fidelity of metagenomic processing methods. *Nature Methods*, 4:495–500, 2007.

[55] KE McElroy, F Luciani, and T Thomas. GemSIM: general, error-model based simulator of next-generation sequencing data. *BMC Genomics*, 13:74, 2012.

[56] M Pignatelli and A Moya. Evaluating the fidelity of de novo short read metagenomic assembly using simulated data. *PLoS One*, 6(5):e19984, 2011.

[57] V Pittet, E Ewen, B.R Bushell, and B Ziola. Genome sequence of *Lactobacillus rhamnosus* ATCC 8530. *J. Bacteriol.*, 194(3):726, 2012.

[58] V Pittet, TG Phister, and B Ziola. Transcriptome Sequence and Plasmid Copy Number Analysis of the Brewery Isolate *Pediococcus claussenii* ATCC BAA-344T during Growth in Beer. *PLoS One*, 8(9):e73627, 2013.

[59] R Poretsky, LM Rodriguez-R, C Luo, T Despina, and KT Konstantinidis. Strengths and Limitations of 16S rRNA Gene Amplicon Sequencing in Revealing Temporal Microbial Community Dynamics. *PLoS One*, 9(4):e93827, 2014.

[60] KD Pruitt, GR Brown, SM Hiatt, F Thibaud-Nissen, A Astashyn, O Ermolaeva, CM Farrell, J Hart, MJ Landrum, KM McGarvey, MR Murphy, NA O'Leary, S Pujar, B Rajput, SH Rangwala, LD Riddick, A Shkeda, H Sun, P Tamez, RE Tully, C Wallin, D Webb, J Weber, W Wu, M Dicuccio, P Kitts, DR Maglott, TD Murphy, and JM Ostell. RefSeq: an update on mammalian reference sequences. *Nucleic Acids Res.*, [ePub], 2013.

[61] MA Quail, M Smith, P Coupland, TD Otto, SR Harris, TR Connor, A Bertoni, HP Swerdlow, and Y Gu. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics*, 13(1):341, 2012.

[62] MS Rappe and SJ Giovannoni. The Uncultured Microbial Majority. *Annual Review of Microbiology*, 57:369–394, 2003.

[63] DC Richter, F Ott, AF Auch, R Schmid, and DH Huson. MetaSim - A sequencing simulator for genomics and metagenomics. *PLoS One*, 3(10):e3373, 2008.

[64] M Ronaghi, M Uhlen, and P Nyren. A Sequencing Method Based on Real-Time Pyrophosphate. *Science*, 281(5375):363–365, 1998.

[65] MG Ross, C Russ, M Costello, A Hollinger, NJ Lennon, R Hegarty, C Nusbaum, and DB Jaffe. Characterizing and measuring bias in sequencing data. *Genome Biology*, 14:R51, 2013.

[66] N Rusk. Torrents of sequence. *Nature Methods*, 8(44):doi:10.1038/nmeth.f.330, 2011.

[67] F Sanger, G Air, BG Barrell, NL Brown, AR Coulson, JD Fiddes, CA Hutchison, PM Slocombe, and M Smith. Nucleotide sequence of bacteriophage Phi-X174 DNA. *Nature*, 265(5596):687–695, 1977.

[68] M Schirmer, L D'amore, N Hall, and C Quince. Error profiles for next generation sequencing technologies. *EMBnet.journal*, 19:81–83, 2013.

[69] CW Schmidt. Data Explosion: Bringing Order to Chaos with Bioinformatics. *Environ. Health Perspect.*, 111:a340–a345, 2003.

[70] R Schmieder and R Edwards. Quality control and preprocessing of metagenomic datasets. *Bioinformatics*, 27:863–864, 2011.

[71] N Segata, L Waldron, A Ballarini, V Narasimhan, O Jousson, and C Huttenhower. Metagenomic microbial community profiling using clade-specific marker genes. *Nature Methods*, 8:811–814, 2012.

[72] J St. John. `http://github.com/jstjohn/SimSeq/`, 2012. [Online, accessed May 25, 2014].

[73] J St. John. `http://github.com/jstjohn/SimSeq/issues/4`, 2012. [Online, accessed May 25, 2014].

[74] M Susman. Genes: Definition and Structure. *eLS*, 2014. doi:10.1002/9780470015902.a0001494.pub3.

[75] CM Thomas and D Summers. Bacterial Plasmids. *eLS*, 2008. doi:10.1002/9780470015902.a0000468.pub2.

[76] J Thompson, F Plewniak, and O Poch. BAliBase: A benchmark alignments database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, 15:87–88, 1999.

[77] B Trost. Evaluation of database search methods for the functional and taxonomic analysis of wgs reads, 2013. Unpublished technical report, Department of Computer Science, University of Saskatchewan.

[78] H Tuomisto. A consistent terminology for quantifying species diversity? Yes, it does exist. *Oecologia*, 4:853860, 2010.

[79] RL Waren, GG Sutton, SJ Jones, and RA Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, 2007.

[80] JD Watson and FH Crick. Molecular structure of nucleic acids. *Nature*, 171(4536):737–738, 1953.

[81] J Wooley, A Godzik, and I Friedberg. A Primer on Metagenomics. *PLoS Comp. Biol.*, 6(2):e1000667, 2010.

[82] AZ Worden, J Janouskovec, D McRose, A Engman, and RM Welsh. Global distribution of a wild alga revealed by targeted metagenomics. *Current Biology*, 22(17):R682–R683, 2012.

[83] M Xu, D Fujita, and N Hanagata. Perspectives and challenges of emerging single-molecule DNA sequencing technologies. *Small.*, 5(23):2638–49, 2009.

[84] D R Zerbino and E Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[85] Y Zhao, H Tang, and Y Ye. RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. *Bioinformatics*, 28(1):125–6, 2012.