Fall 12-1-2017

Development, Evaluation, and Application of a Novel Error Correction Method for Next Generation Sequencing Data

Isaac Akogwu
University of Southern Mississippi

DEVELOPMENT, EVALUATION, AND APPLICATION OF A NOVEL ERROR CORRECTION METHOD FOR NEXT GENERATION SEQUENCING DATA

by

Isaac Onoja Akogwu

A Dissertation
Submitted to the Graduate School,
the College of Science and Technology,
and the School of Computing
at The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

December 2017

DEVELOPMENT, EVALUATION, AND APPLICATION OF A NOVEL ERROR CORRECTION METHOD FOR NEXT GENERATION SEQUENCING DATA

by Isaac Onoja Akogwu

December 2017

Approved by:

ABSTRACT

DEVELOPMENT, EVALUATION, AND APPLICATION OF A NOVEL ERROR CORRECTION METHOD FOR NEXT GENERATION SEQUENCING DATA

by Isaac Onoja Akogwu

December 2017

Tremendous evolvement in sequencing technologies and the vast availability of data due to decreasing cost of Next-Generation-Sequencing (NGS) has availed scientists the opportunity to address a wide variety of evolutionary and biological issues. NGS uses massively parallel technology to accelerate the process at the expense of accuracy and read length in comparison to earlier Sanger methods. Therefore, computational limitations exist in how much analysis and information can be gleaned from the data without performing some form of error correction.

Error correction process is laborious and consumes a lot of computational resources. Despite the existence of many NGS data error correction methods, the false positive rate of correction is still quite high while the amount of computational resources consumed is not declining even with improved algorithms. Until now, many error correction algorithms still use bloom filter as their underlying data structure and a comprehensive downstream analysis of a novel organism upon error correction does not currently exist.

With Illumina sequencing being the most popular and most widely used sequencing technique, this dissertation focuses mostly on correcting Illumina based data. We first describe the characteristics of errors in NGS data and the algorithms implemented so far in mitigating these errors. A methodology was presented to investigate error correction given a range of both real and experimental NGS data with specific attention to substitution, insertion, and deletion errors

Secondly, a comprehensive comparative and statistical comparison of these error correction methods was conducted to discern the effects of NGS data properties like genome size, read length, genome coverage depth and correction algorithm on the number of errors that can be corrected. Based on the results of our investigation, we developed a web based workflow called BECOW, a Bioinformatics Error Correction Workflow, which will allow error correction of NGS data over the internet without the need for prior knowledge of command line language.

Third, a novel error correction algorithm, Cuckoo Filter-based Error Correction of Next-generation Data (CECOND), with cuckoo filter as its underlying data structure, was then introduced. Cuckoo filter is based on cuckoo hash table used to dynamically test approximate set membership in O (1) time. By storing items fingerprints, space is maximized leading to a reduction in computational resource consumption. It also results in low false positive (>3%) rates, better than >4% reported by existing methods, are obtained after error correction.

Finally, error corrected timber rattlesnake (Crotalus horridus) data was used to generate de novo draft genome assembly and compared with those generated using other methods. The assembly comparison results proved that error corrected data is desired for qualitative draft genome assembly to be achieved.

ACKNOWLEDGMENTS

These last five years as a graduate student has allowed me to live, interact and work with several great people and researchers without whose support, advice, and help, I would not have completed this dissertation.

Primarily, I am extremely thankful to my supervisor and committee chair, Professor Chaoyang Zhang, who has influenced every aspect of my research. He taught me how and what it takes to be an exceptional researcher. The weekly meetings were essential in teaching me rigorous research ethics and ensured I had the necessary interactions, to be the researcher I wanted to be. I would also like to thank Dr. Ping Gong, of the United States Army Corp of Engineers, for being a great collaborator and for his immense guidance and support. Dr. Ping Gong is a great mentor whose deep insight and experience in working with genomics data inspired the ideas of the research presented in this dissertation.

I was fortunate to have some of the best minds in the field as my supervisory committee members. I would like to express my gratitude to: Dr. Alex Flynt of USM Biological science department, who gave me insights on genome analysis, in addition to his accommodation of my numerous questions. Dr. Nan Wang, Dr. Zheng Wang, and Dr. Wonryull Koh for their feedback, support and general advice.

My sincere gratitude to our collaborators Dr. Douglas Rhoads, Dr. Shane Sanders, and other participants from Mississippi State University, University of Arkansas, and King Abdullah University of Science and Technology, for supplying the timber rattlesnake data evaluated in this work.

Finally, my life as a PhD would not have been more complete without the presence of my fellow lab mates: Andrew Maxwell, Gabriel Idakwo, Mensheng Zha, and all the visiting scholars like, Dr. Runzhi Lee. The ideas, input to my research and encouragement we shared with each other made my stay in the Data Mining and Bioinformatics lab worthwhile. They not only made the work environment more enjoyable, but also became good company outside work. We have built a lifelong friendship.

DEDICATION

I dedicate this dissertation to my father, late Peter Ojeabo Akogwu and to my sons Maxwell Amran Onoja and Castiel Onoja Akogwu, whose thoughts have kept me going. I am also grateful to my spouse, Romona Akogwu, for her loving support and understanding during this five years – she gave me the strength and confidence to stay focused and determined. I would like to express my profound gratitude to my mom, Rosemary Akogwu, my sisters, brothers, and especially to my brother, Emmanuel Peter Oche, for their never-ending support and encouragement. They have always motivated me to remain curious and productive in my professional and personal life. Finally, and most importantly, I will like to give thanks to God for a gift of life without which none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ABBREVIATIONS

ATP	Adenosine triphosphate
BECOW	Bioinformatics Error Correction Workflow
BF	Bloom Filter
BP	Base Pairs
CECOND	Cuckoo-filter Error Correction of Next-generation Data
CF	Cuckoo Filter
DBG	Double Bricks & Gap
DL-CBF	D-left Counting Bloom Filters
DNA	Deoxyribonucleic Acid
DSK	Disk Streaming of K-mers
ECC	Error Correction Code
ECET	Error Correction and Evaluation Toolkit
FN	False Negative
FP	False Positive
GAII	Genome Analyzer II
GATB	Genome Assembly & Analysis Took Box
HMM	Hidden Markov Model
KMC	K-mer Counter
MASURCA	Maryland Super Read Cabog Assembler
MCBIOS	MidSouth Computational Biology and Bioinformatics Society
MEGA	Molecular Evolutionary Genetics Analysis
MPSS	Massively Parallel Signature Sequencing

MSA	Multiple sequence alignment
NGS	Next-Generation Sequence
NIPT	Non-invasive Prenatal Tests
NT	Nucleotide
OGDRAW	Organellar Genome Draw
OTU	Operational Taxonomic Units
PacBIO	Pacific Biosciences
PCR	Polymerase Chain Reaction
PE	Paired-end
PGD	Preimplantation Genetic Diagnosis
QC	Quality control
QUAST	Quality Assessment Tool for Genome Assemblies
RAM	Random Access Memory
RNA	Ribonucleic Acid
rRNA	ribosomal RNA’s
SBE	Single Brick & Edges
SBL	Sequencing by Ligation
SFF	Standard Flowgram format
SMRT	Single Molecule Real Time
SMS	Single Molecule Sequencer
SNP	Single Nucleotide Polymorphism
SOLiD	Supported Oligonucleotide Ligation Detection
SSPACE	SSAKE Scaffolding of Pre-Assembled Contigs after Extension

TEF	Target Error Format
TGS	Third Generation Sequence
TP	True Positive
TRNA	Transfer Ribonucleic Acid
USM	The University of Southern Mississippi

CHAPTER I - INTRODUCTION

1.1 General Overview

DNA sequence defines the information encoded in an organism. Determining the ordering of the sequence is critical to understanding the information. This is achieved by performing genome sequencing. Despite knowledge of the nature of DNA (Deoxyribonucleic Acid) as an information encoder (from study by Oswald Theodore Avery in 1944), little was known of its structure until it was discovered in 1953 [1]. Even then, no one thought the pace of progress in the field will be this fast. Since 2005, DNA sequencing has evolved leading to NGS (Next-Generation Sequence) technologies which operates at exceptional speeds. NGS has been widely accepted [2] and currently, sequencing on an Illumina machine is 1,000 times cheaper than Sanger with 10,000 more data generated at the same amount of time [3]. Sequencing the genome multiple times (high coverage) has allowed large genomes like that of humans and loblolly pines to be sequenced and assembled. However, assembling the genome of many other species e.g. timber rattlesnake remains difficult due to multiple challenges.

These challenges are due to significantly short read length, incorrectly called bases that lead to errors in the sequence and technical difficulties of using the technologies. Understanding how sequence reads produced by the various technologies differ is paramount for thorough and qualitative analysis of the sequence reads. Challenges of short read length are due to the large size of the genome to be sequenced. Current sequencing machines are unable to sequence the whole genome at once because sequence library preparation methods employ slicing of the full genome before it goes to be read by the sequencers. Characteristics of NGS technologies have been studied in [4] and [5]. Errors in base calling, which involves determining nucleotide ordering from the sample, is one of the most fundamental challenges encountered in NGS technologies.

In this chapter, we first present an explanation to justify the inspiration for the work in this dissertation. Secondly, a summary of the main aim of this dissertation is outlined. Next, specific accomplishments gained from the various methods implemented in our work is presented and finally, we provide a description of the layout of the entire dissertation.

1.2 Research Motivation

Accurate base calling is important for analysis such as genotype calling, genome assembly, SNP (Single Nucleotide Polymorphism) detection, genome resequencing etc. Although it can aid sequencing cost reduction because accurately called bases will require less sequence coverage, having high read coverage is not a guarantee to achieving accurate analysis. These base calling errors create complexity in analysis because it can either prevent identification of overlapping reads or it can include information which can misguide an analysis.

Although, technological advancements have reduced errors to about 1 per 100 BP (Base Pairs) read [6], it is still not acceptable for quality downstream analysis. This makes it extremely paramount to remove all errors. Review of current literature, discussed in CHAPTER II, highlights the existence of many error correction methods developed to alleviate issues that lead to poor downstream analysis. Continuous improvement in the methods are still constantly been sought after. Many of the methods are not space efficient, therefore consumes a lot of computational resources during the error correction process. The error correction methods also exhibit high false positive rates. Furthermore, published error correction methods only use genome assembly quality metrics and contig size to show that their methods work. None of the existing methods have been used to completely assemble the genome of a previously un-sequenced organism after error correction during the evaluation of the method.

1.3 Research Purpose

The main aims of this work are to improve upon existing error correction methods discussed in CHAPTER II and apply the method to characterize the genome of a novel organism. A sequence of steps is necessary to achieve our goal. These steps are each dependent on the knowledge gained from a prior step thereby creating an effective way of understanding the processes even for novice in the field. The steps taken are clustered into four major goals:

1. To provide the ground truth for this work from an NGS error correction perspective. To achieve this aim, a limited description of the history of NGS is first given to acquaint the reader with its progressive nature. A background of the sequencing technologies is also elaborated with an in-depth focus on errors in the sequence data and methods available to correct these errors.

2. To establish a foundation by gaining more insights from the implementation approaches used by existing error correction methods. For improvement to be made over existing methods, it is imperative to understand their underlying structure and the differences between them. This can be achieved by performing a comparative and statistical analysis to measure their performance on a varied number of NGS datasets. Measuring the performance of the methods will allow us to make

recommendations to users on why and when to use specific methods and provide a more user-friendly way of using the methods.

3. To develop a novel error correction method if it is discovered, from earlier step, that limitations exist in the current methods employed for error correction. Identification of any associated problems will allow the developed method to be more robust to alleviating the discovered issues. In addition, make a comparison of the novel and existing methods to see if the issues have been resolved.

4. To use the developed error correction method to correct the sequence data of a previously un-sequenced organism. This will help in the evaluation of the novel data in a real experiment without an existing reference genome.

The set goals for this dissertation are organized in a methodological manner to be accessible and used, not just by seasoned molecular or computational biologists, but by a wide range of audiences.

1.4 Contributions

This dissertation examines and confronts the above-mentioned challenges in terms of algorithm efficiency, computational resource consumption, method evaluation and application. All the goals outlined were successfully addressed in the following chapters of this dissertation.

First, genomic sequencing was addressed based on its history and progress over the years. We proceed to define the type of base calling errors that exist or are prevalent in the different types of NGS platforms. Secondly, with the current significant availability and popularity of data from Illumina based technologies, focus was specifically placed on studying errors generated by that technology and the algorithms capable of resolving those errors. Upon complete statistical analysis and evaluation of the chosen NGS error correction methods, their shortcomings were presented.

A novel method known as CECOND (Cuckoo-filter Error Correction of Next-generation Data), based on Cuckoo filter data structure and a combination of error correction steps, was proposed and implemented. CECOND is categorized as a k-mer spectrum based error correction method implemented to solve the shortcomings of existing k-mer spectrum based error correction methods. The results were analyzed using statistical methods and alignment. Finally, the method was used to correct Illumina data of timber rattlesnake and the results were evaluated by a comparative assessment in relation to an existing draft genome assembly of the rattlesnake.

Each of the goals of this research has been achieved as individual chapters in this dissertation.

1.5 Works Presented and Published

Part of this dissertation have been published as peer reviewed journal publication, conference publication while some were presented as oral and poster presentations at various conferences and symposiums.

The whole of CHAPTER III was published or currently in review under the following:

1. Influence of Illumina Sequencing Dataset Characteristics on Performance of Error Correction Tools. Submitted and in review by the journal: Nature Scientific Reports on September 27, 2017

2. A comparative Study of K-spectrum-based Error Correction Methods for Next Generation Sequencing Data Analysis. This was published in Vol 10 Supplement 2 of Human Genomics, 2016
3. Factorial analysis of error correction performance using simulated next-generation sequencing data. Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference Proceeding

The works in CHAPTER III and CHAPTER IV were also presented as poster and oral presentations at conferences and symposiums under the following:

1. Becow: A Web-Based Bioinformatics Error Correction Workflow Tool for Next Generation Sequence Data Correction at the XIII MCBIOS Annual Conference, March 3-5, 2016, Memphis, TN (Poster Presentation)
2. An Integrated Statistical Probe of Next Generation Sequencing Error Correction Frameworks for Illumina DNA Sequence Data at the XIII MCBIOS Annual Conference, March 3-5, 2016, Memphis, TN (Oral Presentation)
3. A Comparative Study of K-mer-spectrum Based Error Correction Methods for Next-Generation Sequencing Data Analysis at the XII MCBIOS Annual Conference, March 12-14, 2015, Little Rock, AR (Oral Presentation)
4. De Novo Assembly and Functional Annotation of Timber Rattlesnake (*Crotalus horridus*) Genome from Next Generation Sequence Data at the XI MCBIOS XI Annual Conference, March 6-8, 2014, Stillwater, OK (Oral Presentation)

1.6 Organization of This Dissertation

The organization of this dissertation continues as follows: CHAPTER II discusses the progress of DNA sequencing over the years, the different types of sequencing technologies, their drawback, and the types of errors they contain. Also, a biological background for the research is presented here. CHAPTER III presents the various comparative and statistical evaluation of error correction methods. The contents of this chapter have already been published but further elaboration have been provided here to support our claims. CHAPTER IV proceeds with introducing BECOW (Bioinformatics Error COrrection Workflow), previously unpublished. BECOW is one of the results of the evaluations conducted in the previous chapter. CHAPTER V presents the novel error correction method called CECOND with a look at its performance through a series of evaluations using both simulated and experimental data. We presented its error correction model and evaluated its computational resource consumption in comparison to exiting methods. CHAPTER VI discusses the implication of error correction for genome assembly by application to timber rattlesnakes. Finally, a conclusion describing the general contributions of the research and future directions was presented in CHAPTER VII.

CHAPTER II – GENOME SEQUENCING AND ERRORS

2.1 Chapter Overview

Genome sequencing has revolutionized how biological experiments are conducted to unravel DNA. Its contribution to different fields of molecular biology including medicine, forensics, agriculture, and a host of other applications are visibly apparent. Here, the progress of DNA sequencing over the years, the different types of sequencing technologies, their drawbacks, and the types of errors they contain are first discussed. Secondly, NGS errors, their detection, correction and overall impact of the corrections are elaborated. Thirdly, we discuss the various types of error correction algorithms including their classification and applicable data structures. Furthermore, related existing error correction methods like Bloocoo, BLESS, BFC, Musket, Lighter and Trowel which are all kmer-spectrum based are discussed. We basically focused on parameter selection, types of errors they can handle and the types of NGS data they are most suitable for. Finally, the chapter ends with a discussion on the drawbacks of the error correction methods and systematically proposes a solution to reduce, if not eliminate, the problems associated with the existing methods.

2.2 Genomic DNA Sequencing

Understanding DNA (Deoxyribonucleic acid) sequencing entails looking at a brief history of how the technologies for studying nucleic acids evolved from DNA discovery until now Figure 2.1. Aside using DNA to decode the life mysteries surrounding living organisms, it provides evidences about the environment, origin, phylogeny, disease vulnerability of an organism. Decoding the DNA constitute a paramount challenge for scientist and has led to constant evolvement of sequencing platforms. Each technology improves based on prior discoveries as they evolve. These improvements over the years, have been grouped into various contentious categories based on sequencing method [7] or suggested differences between 2nd and 3rd generation sequencing [8][9][10][11] with some methods existing within the boundary of both suggestions.

Classification referred to in this dissertation is based on the period of release because in most cases, a newer sequencing platform improves upon an existing technology. We have grouped the sequencers as Sanger, Next, Third and Fourth generation sequencing, in increasing order of development.

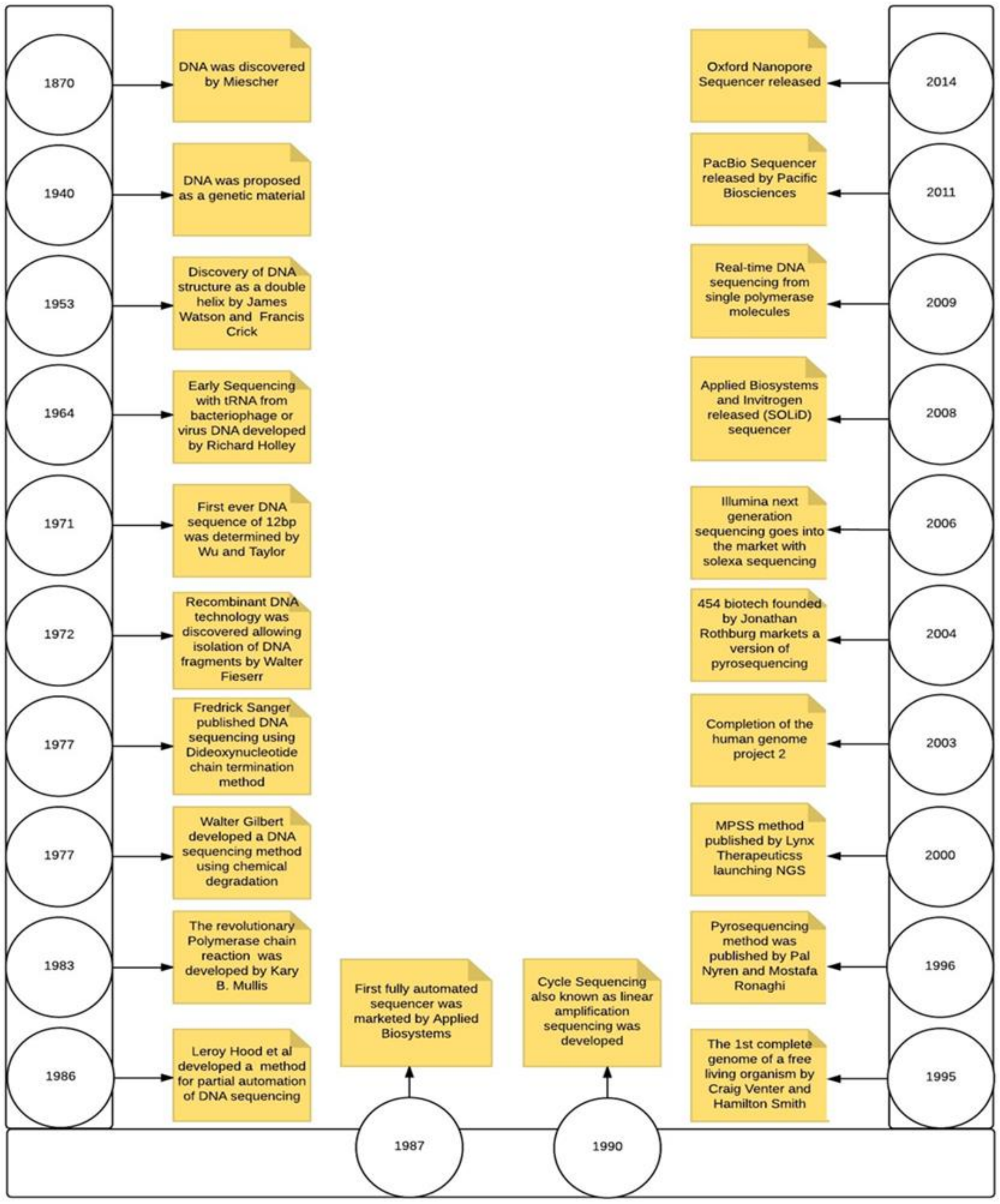

1870
DNA was discovered by Miescher
1940
DNA was proposed as a genetic material
1953
Discovery of DNA structure as a double helix by James Watson and Francis Crick
1964
Early Sequencing with tRNA from bacteriophage or virus DNA developed by Richard Holley
1971
First ever DNA sequence of 12bp was determined by Wu and Taylor
1972
Recombinant DNA technology was discovered allowing isolation of DNA fragments by Walter Fieserr
1977
Fredrick Sanger published DNA sequencing using Dideoxynucleotide chain termination method
1977
Walter Gilbert developed a DNA sequencing method using chemical degradation
1983
The revolutionary Polymerase chain reaction was developed by Kary B. Mullis
1986
Leroy Hood et al developed a method for partial automation of DNA sequencing
1987
First fully automated sequencer was marketed by Applied Biosystems
1990
Cycle Sequencing also known as linear amplification sequencing was developed
1995
The 1st complete genome of a free living organism by Craig Venter and Hamilton Smith
1996
Pyrosequencing method was published by Pal Nyren and Mostafa Ronaghi
2000
MPSS method published by Lynx Therapeuticss launching NGS
2003
Completion of the human genome project 2
2004
454 biotech founded by Jonathan Rothburg markets a version of pyrosequencing
2006
Illumina next generation sequencing goes into the market with solexa sequencing
2008
Applied Biosystems and Invitrogen released (SOLiD) sequencer
2009
Real-time DNA sequencing from single polymerase molecules
2011
PacBio Sequencer released by Pacific Biosciences
2014
Oxford Nanopore Sequencer released

Figure 2.1 Summary of the history of DNA sequencing discovery and how it has evolved to the most recent times. The circles indicate the year while the yellow boxes indicate the sequencing events that occurred within that year. Details extracted from various sources.

2.3 Brief history of DNA sequencing Discovery

Long before 1944, when DNA was assumed to be first known as the genetic material, the first purification of DNA was obtained by Friedrich Miescher in 1870. The work of Avery et. al. led to James D. Watson and Francis Crick [1] discovering that the double helix strand structure contains four bases in 1953. Having proved it is the hereditary material and the structure determined, the focus of biologists shifted from studying whole organisms to their component cells. Determining the nucleotide ordering in biological samples became an integral component of several research applications. The first homogenous purification of DNA molecule was the genome of bacteriophage ϕX174 in 1959 [12]. In 1964, the earliest known sequencing was performed with TRNA (Transfer ribonucleic acid) from Bacteriophage [13]. It involved breaking down and piecing back together the pieces of RNA molecules. The process was laborious and took too long to sequence because of the large DNA size. This resulted in using primer extension [14][15][16] as the first method for DNA sequencing. They reported a partial sequence in 1968, but successfully sequenced 12bp cohesive ends of phage l DNA by 1971 [17].

This discovery opened opportunities for DNA sequencing but due to its application only to short sections of lambda phage genome ends, oligonucleotide primers in DNA sequencing reactions [18] was introduced to generalize the process. Subsequently, synthetic primers that binds to specialized locations were used from 1970-1973. Current sequencing methods resulted from the discovery of type II enzymes which binds to DNA at definitive lengths of 4bp-6bp without the need for ATP (Adenosine triphosphate) during degradation [19][20][21]. This led to the realization that screening bacterial strains [22] resulted in sequence recognizing enzymes known as restriction enzymes. These enzymes Figure 2.2 therefore, provided a general method for DNA fragmentation which are subsequently separated using gel electrophoresis.

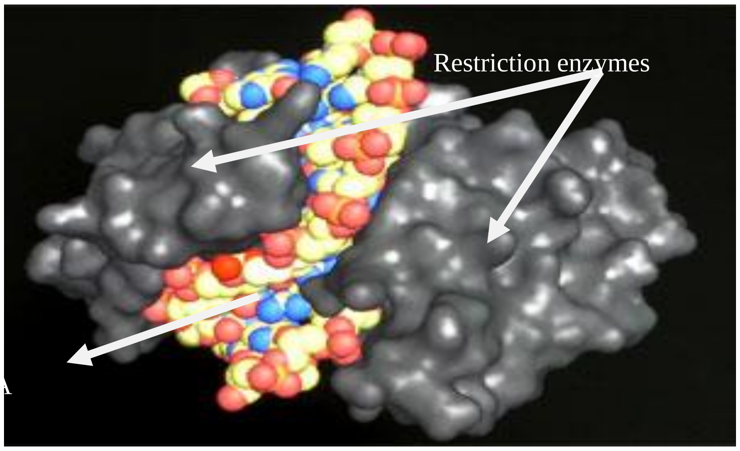


Figure 2.2 Restriction enzyme Mva1 (grey) is shown wrapped around DNA (multicolored) [23]. Protein database ID: 2OAA. New England Biolabs

2.3.1 First Generation Sequencing

Earlier modern methods used base specific chemical reactions (depurination) like those used for RNA (Ribonucleic Acid) sequencing. They applied separation methods, to limit the size of the fragments. Fredrick Sanger introduced the plus and minus approach [24] which became the pioneer method for modern sequencing technologies. It used chain termination principle and was used to sequence a complete genome sequence of the ϕX DNA genome [25]. Despite this result, better methodology was still been sought after by Sanger [26] which led to a chemical degradation method like the plus and minus method [27]. The only difference being that it produced bands for every sequence position. This advantage led to its early adoption until Sanger realized that copying the DNA was better than degrading it [28]. The method became a basis for subsequent technologies Figure 2.3.

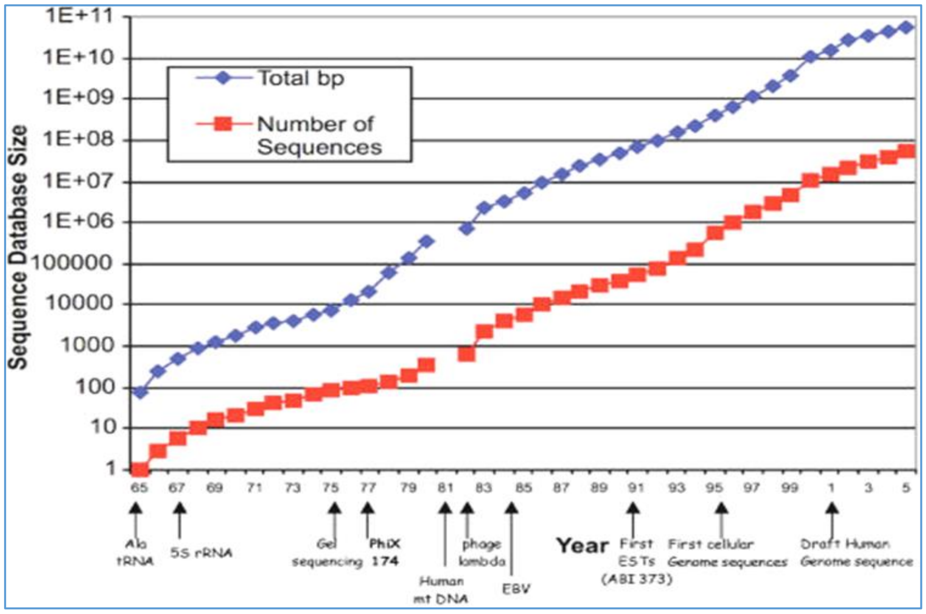


Figure 2.3 Growth of the nucleotide sequence database. The number of published nucleotide sequences, and the total number of base pairs of sequence are plotted versus the date of deposition or publication. Data since 1981 are re-plotted from http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html and data for sequences published before 1981 are from Dayhoff, Nucleic Acid Sequence Database, Vol. 1. The dates of landmark sequences and technological advances are indicated. Adapted from [29].

The revolutionary PCR (Polymerase Chain Reaction) [30] was developed in 1983 whereby a single DNA gets exponentially amplified in a selective way to generate multiple copies of the DNA. Automation of the process led to the first partial automation method of DNA sequencing in 1986 [31] through a modification of the chemistry and data gathering method of the Sanger method. Regardless, there was a need for a faster and inexpensive sequencing which led to pyrosequencing or sequencing by synthesis.

Pyrosequencing published in 1996 [32][33], as opposed to Sanger sequencing, is based on how much pyrophosphate released can be detected upon fusion with nucleotide instead of chain termination [24]. It works in real-time to monitor the pyrophosphate released allowing the DNA sequence to be determined. Increasing the read length is among its greatest disadvantages resulting in Next generation of DNA sequencing.

2.3.2 Next-Generation Sequencing Technologies

Competition for the fastest and most efficient sequencing technologies promoted rapid advances in new methods. A concept presented in [34] led to MPSS (Massively Parallel Signature Sequencing) in 2000. This method used parallelized adapter bead based sequencing of four nucleotides at a time. The technology was so complex and susceptible to sequence bias. MPSS became obsolete when a sequencing by synthesis was developed. However, the important properties of MPSS output made up parts of later NGS data types. Lynx Therapeutics merged with Solexa in 2004 and later purchased by Illumina.

2.3.2.1 Polony Sequencing

Polony, an acronym for Polymerase-colony sequencing developed at Harvard and published in 2003 [35] was one of the first Next Generation Sequencing technology with an open-source platform. Polony is a non-electrophoretic SBL (Sequencing By Ligation) method. i.e. it uses short DNA segments called oligonucleotides and not single bases for sequencing Figure 2.4 It contains three (3) replaceable major steps: library construction, emulsion PCR and sequence data. Polony was not commercially successful, even though it is easy to use, because of the cost, short reads, and ligation time. Although it was used to sequence a full genome of *Escherichia coli* in 2005 with an accuracy almost 100%.

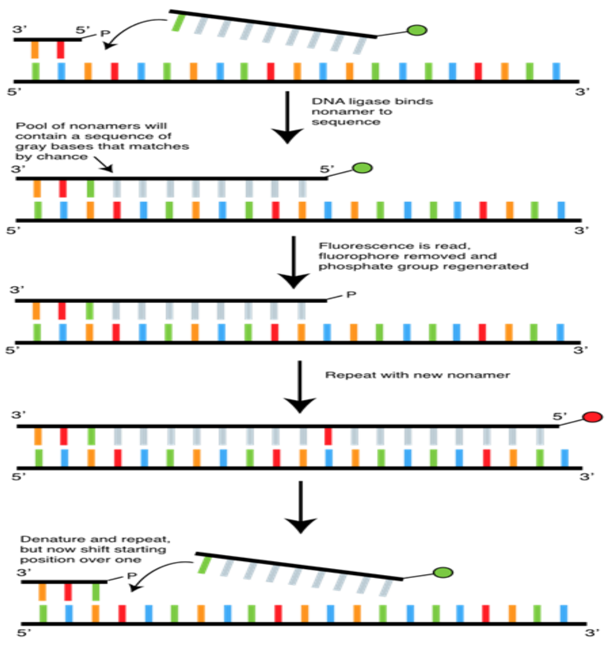

3'
5'
P
5'
3'
DNA ligase binds nonamer to sequence
Pool of nonamers will contain a sequence of gray bases that matches by chance
Fluorescence is read, fluorophore removed and phosphate group regenerated
Repeat with new nonamer
Denature and repeat, but now shift starting position over one

Figure 2.4 How sequencing by ligation works adapted from [36]

2.3.2.2 Roche 454 Technology

With the potentials of pyrosequencing, 454 life science decided to develop a version of pyrosequencing that was parallelized. The pyrosequencing method, Figure 2.5, in their instrument employed sequencing by synthesis and became the first commercialized next generation sequence technology. The sequences are determined while the complementary strand is being formed as opposed to polony which is ligation based, Figure 2.6. However, its reliance on the detection of pyrophosphate molecules makes it like polony. The amount of pyrophosphate released by DNA polymerase equals the amount of nucleotide added thereby initiating reactions that generate the light [37] detected by a sensor. Currently in 2016, the read length is about 1kb. The speed and read length gives the Roche system a competitive edge over other NGS methods. The drawback for Roche 454 remains the high reagent cost and the relatively high amount of indel errors.

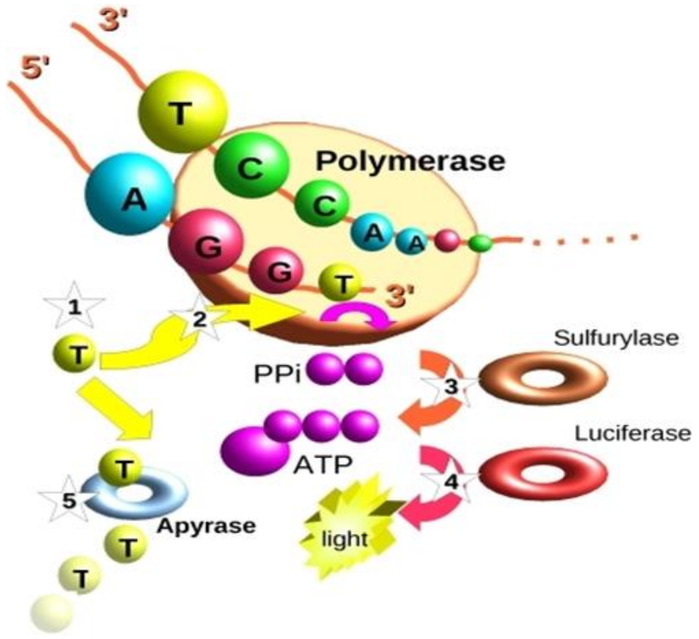


Figure 2.5 shows how pyrosequencing works. Pyrosequencing. Single stranded DNA template is first hybridized with the sequencing primer and mixed with the enzymes along with the two substrates adenosine 5′-phosphosulfate (APS) and luciferin. In each cycle, (1) one of the four nucleotides (dTTPi, in this case) is then added to the reaction. (2) If the nucleotide is complementary to the base in the template strand then the DNA polymerase incorporates it into the growing strand. (3) Pyrophosphate (PPi)—in an amount equal in molarity to that of the incorporated nucleotide—is released and converted to ATP by sulfurylase in the presence of APS. (4) ATP then serves as a substrate to luciferase, causing a light reaction. Photon emission is in equimolar quanta to the amount of nucleotide incorporated in each cycle. (5) apyrase degrades the excess nucleotides. Figure was adapted from [38]

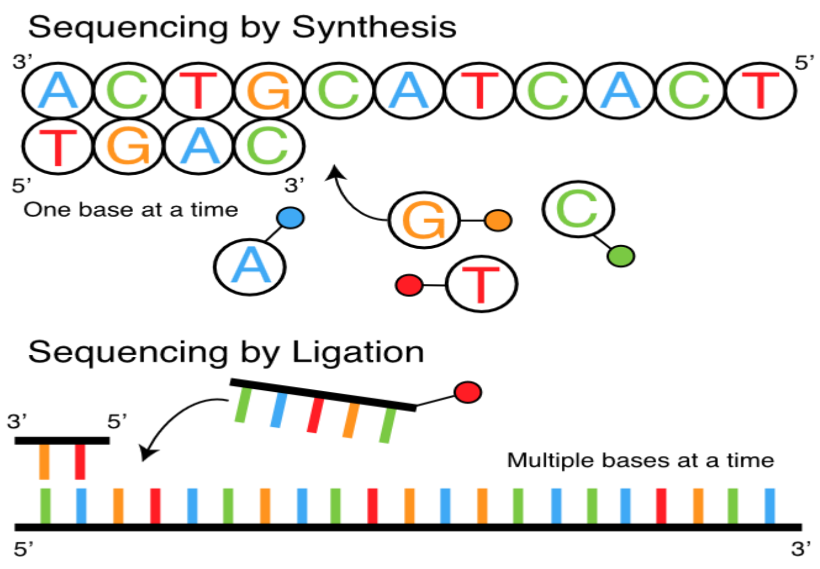


Figure 2.6 Difference between sequencing by synthesis vs. sequencing by ligation adapted from [36]

2.3.2.3 Illumina (Solexa) Sequencing

Illumina sequencing released in 2006 was the second of the NGS technology in the market. It is based off Solexa sequencing. It uses sequencing-by-synthesis strategy, a chain termination based method known as reversible termination sequencing [39]. It tracks the addition of labelled nucleotides as the DNA chain is copied. It's ability to reversibly terminate primer extension due to fluorescent label on the terminating base as opposed to irreversible termination in Sanger marks a significant difference [40]. Since base reading steps are separated from each other in a homopolymer run, Illumina also does not generate a lot of indels like 454. It's system of high densities clusters that can be analyzed, allows it to generate data ranging from 300 KB up to 1 TB in a single run. Signal deterioration due to the incorporated reversible dye terminator nucleotides or cleavage of fluorescent labels causes its read length to be shorter than that of 454. The presence of this dye terminator and use of modified polymerase nucleotides also makes base-substitution errors as the dominant error type. Average raw error-rates are about 1–1.5%, but higher accuracy bases with error rates of 0.1% or less can be identified through quality metrics associated with each base-call. As with other systems, modifications have recently enabled mate-paired reads; for example, each sequencing feature yielding 2 × 36 bp independent reads derived from each end of a given library molecule several hundred bases in length. Illuimina process is shown in Figure 2.7

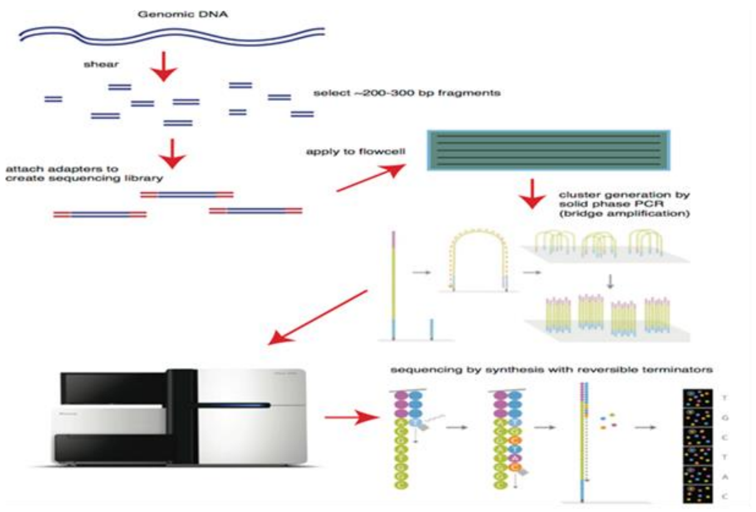

Genomic DNA
shear
select ~200-300 bp fragments
attach adapters to create sequencing library
apply to flowcell
cluster generation by solid phase PCR (bridge amplification)
sequencing by synthesis with reversible terminators

Figure 2.7 Process of generating Illumina sequencing data [41]

2.3.2.4 ABI SOLiD Sequencing Technology

In 2008, the SOLiD (Supported Oligonucleotide Ligation Detection) system, based on a hybridization-ligation chemistry [42], was released. It is a massively parallel sequencer like 454 in principle but differs in terms of the bead size and the random array format used. Colonal bead populations are obtained from DNA fragments of sequencing samples. These fragments, in addition to a mixture of sequencing primers and fluorescently labelled probes go through repeated process of hybridization and ligation. The encoded base on the probes are detected based on the perceived fluorescent signals. Multiple cycles of the entire process are performed with the read length determined by the number of cycles. Although the read length is short, $25 - 35$, it can generate up to 3 GB of sequence per run. The final error rates for SOLiD is <= 0.1%

2.3.3 Third-Generation Sequencing

Although a huge amount of data is generated by second generation sequencers at consistently lower cost per base, important properties like read length, cost per run and time to completion have been overlooked. The goal of Third generation sequencing is to solve these setbacks. In this dissertation, we consider third generation technologies to be those capable of sequencing single molecules without amplification.

2.3.3.1 Helicos SMS Technology

Helicos BioSciences commercialized the first Single Molecule Sequencer (SMS) technology developed by Stephen Quake [43][44]. It uses the same SBS method as Illumina does, but without any amplification. It performs True Single Molecule Sequencing tSMS chemistry to observe the SBS reaction of individual DNA molecules in parallel. Single fragmented DNA molecules are combined into single DNA strands and then deposited to a Flow Cell glass surface. Using a propriety fluorescent reversible virtual terminator dNTPs [45], the nucleic acid bases are identified one base at a time by capturing the image and its cycle information. While relatively slow and expensive, this was the first technology to allow sequencing of non-amplified DNA.

2.3.3.2 PACBIO single molecule real time (SMRT)

Pacific Biosciences developed the SMRT platform. The pacBio SMRT method unlike NGS allows real time DNA sequencing without pausing between read steps. It exploits the properties of light passing through holes of a diameter smaller than its wavelength thereby allowing only a single immobilized DNA polymerase or template at the bottom of a well to be illuminated. Because sequencing occurs at the rate of the polymerase, it produces kinetic data, which leads to detection of modified bases [46].

PacBio machines can produce long reads exceeding 10 kb in length offering much longer lengths and faster runs than NGS methods but is mired by a lower throughput, higher raw read error rate >5% and higher cost per base when compared to NGS or tSMS.

2.3.4 Fourth generation Sequencing

Fourth generation sequencing refers to sequencing methods that neither require a DNA amplification step nor use any expensive reaction agents as part of library preparation. DNA sequencing by other generation technologies has become increasingly faster and cheaper but most of them either use fluorescent reagents to identify nucleotides or require chopping up the DNA molecule and amplifying the fragments.

2.3.4.1 Oxford Nanopore Technology

Oxford Nanopore sequencer was the first and only currently known fourth generation of sequencers available. As the name implies, it is based on nanopores. A nanopore is a 1 nanometer diameter hole, only capable of allowing a single DNA molecule to thread through it at a time. The idea behind the nanopore sequencers is that, an electrically charged fluid can create a specific change in the amount of current in an immersed nanopore when single nucleotides (bases) of a DNA, or other molecules pass through or near the nanopore. Specific change in current corresponding to each base passing through the pore is measured to decode the entire order of the DNA sequence Figure 2.8. There is also the potential to use solid-state technology to generate suitable nanopores, allowing sequencing of double stranded DNA molecules [47][48]. The GridION was the first, then came the MinION, which is a USB sized device, first tried by users in 2014 [49]. Then PromethION which is compatible with cloud based services.

SmidgION, the smallest sequencer ever that can be plugged into a smartphone for sequencing producing 230 Mb/hour is expected in late 2017 [62]

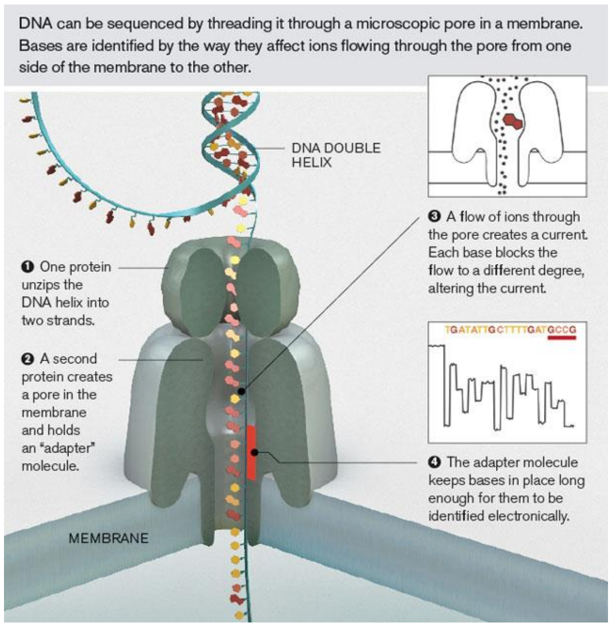

DNA can be sequenced by threading it through a microscopic pore in a membrane. Bases are identified by the way they affect ions flowing through the pore from one side of the membrane to the other.
DNA DOUBLE HELIX
❶ One protein unzips the DNA helix into two strands.
❷ A second protein creates a pore in the membrane and holds an "adapter" molecule.
❸ A flow of ions through the pore creates a current. Each base blocks the flow to a different degree, altering the current.
TGATATTGCTTTTGATGCCG
❹ The adapter molecule keeps bases in place long enough for them to be identified electronically.
MEMBRANE

Figure 2.8 Nanopore DNA sequencing as employed in ONT's MinION sequencer. Double stranded DNA gets denatured by a processive enzyme (†) which ratchets one of the strands through a biological nanopore (‡) embedded in a synthetic membrane, across which a voltage is applied. As the ssDNA passes through the nanopore the different bases prevent ionic flow in a distinctive manner, allowing the sequence of the molecule to be inferred by monitoring the current at each channel

2.4 Significance of Sequencing Projects

Sequencing projects are performed for a variety of reasons. DNA sequencing as mentioned in 1.1 involves determining the ordering of the nucleotide sequence of a given DNA, the genetic material of most organisms. This implies defining the exact order in which the bases; adenine, cytosine, thymine, and guanine, occur. Sequencing is very complex and has gone through several modifications technology-wise. Just sequencing the DNA is by itself, not enough. Sequencing projects are performed to answer specific biological questions. There are currently several ongoing sequencing projects like the human genome project, the plant genome project, etc. The significance of these projects is enormous and cannot be completely stated in this dissertation. This is because new applications are being sought as the technologies improve. Projects like the human genome projects have potential significance for:

- Molecular medicine – for disease diagnosis and discovery where proper identification of a gene may lead to accurate diagnosis. It can also lead to detection, for risk assessment, of an individual's predisposition to certain genetic disorders like cancer, metabolic and heart diseases. Furthermore, it benefits targeted drug design for gene products which lead to diseases while also allowing performance of gene and drug therapies. In these cases, the genes are reengineered by replacing defective disease-causing genes and drug administration based on a person's genotype respectively. Also aids Preimplantation Genetic Diagnosis(PGD) and Non-invasive Prenatal Tests (NIPT) whereby embryos can be tested for genetic characteristics and early genetic screening for chromosomal conditions. It also aids preconception and preimplantation screening to enable informed choices about reproductive status. Finally, it allows exploration of genetic makeup of an individual to determine their response to medications in pharmacogenomics.

- Forensics – determining the paternity of a child and other family relationships, identifying crime suspects and victims based on DNA evidences e.g. hair and blood or even to absolve individuals wrongfully accused of a crime. It can also aid missing person's cases and identification of disaster victims

- Evolution – study population migration based on genetic inheritance, evolution through germline mutations in lineages, etc.

Projects like the Floral, plant and animal genome projects are more complex due to the presence of multiple chromosome copies and repetitive sequences, yet, like their human genome counterpart, once sequenced is applicable to:

- Agriculture – Using conserved genes to determine the evolutionary history of plants, understanding plant to pathogen and plant to insect relationships, Identification of micro-organism that may endanger crop and plant growth or productivity through air, soil, or water pollution, engineering crops to produce better quality products with increased nutrient value, improve resistance to insects and pests. In addition, even plants are being reengineered to carry edible vaccines for certain human diseases allowing the antigens to exist in the cell of the plants. Finally, is it also used for the identification of both endangered and protected wild life species,

The above significance of sequencing a genome is nowhere near complete hence, there are many more applications known and yet to be determined.

2.5 Characteristics of Sequencing Data

Genome sequencing data, be it de novo sequencing, genome resequencing or transcriptome sequencing, possess important characteristics that aids its analysis and understanding. These characteristics are unique to each set of sequenced data and are determined by the sequencer, sequencing platform or technology used and the organism been sequenced. Here, a brief explanation of some of the major characteristics are elaborated with a focus on NGS data. Genome Sequence Coverage

2.5.1 Genome Sequence Coverage

Often, the concept of depth of coverage gets misused or misunderstood. The coverage gives an average measurement taken for each base of the genome been sequenced. i.e. each base is sequenced a coverage number of times. Sequence coverage is a key consideration when performing genome analysis since it determines the amount of data generated and depends on the type of questions been investigated by the experiment. The error rate of sequencing technology, the complexity of repeats of the genome under consideration and the read length determines the coverage depth of sequencing required. Sequencing to high depths does not necessarily improve certain downstream analysis like assembly [50]. The average coverage (X) for a genome (G) with number of reads (N) can be calculated using various formulae depending on the read Length (L) as follows:

1. For equal read length: $X = N*L / G$
2. For variable length L of read j: $X = \text{sum}(L_j) / G$

Since DNA fragmentation from sequencer exhibits countless non-definitive patterns due to bias which cannot be easily modelled, theoretical coverage is not ideal for practical purposes. For these reasons, the average coverage is used. Been that this coverage is only average for a random process, it is certain that variations exist from one base position to another and some bases may not have been covered. The probability of a base not getting covered or sequenced can be calculated based on the Lander/Waterman theoretical model of random fragmentation as: Probability (p) = exponential value of (-X), again X been coverage. This gives the number of missing bases in the sequenced data. High coverage depth is desired because low depth introduces sequence errors that may propagate through downstream analysis and lead to inaccurate conclusions during an analysis.

2.5.2 Read Length

Sequencing a genome is performed with an initial idea or goal of stitching the fragments to obtain the draft genome. The fragments generated from sequencing flow cells are often of a given length. This length is known as the sequence read (sequence information) length. It is the number of bases ACTG sequenced. Read length is sometimes referred to as cycles. On Illumina platforms, using e.g. 200 and 250 cycles for base pair sequencing will result in two 200bp and two 250bp read lengths respectively from one piece of DNA. These pair are separated by a length (insert size) chosen during material preparation for the sequencing project Figure 2.9. Often, depending on the experiment, longer read lengths are desired but read length can only be configurable up to a maximum length. This implies that it is sequencer platform dependent. For example, the read length is specified by the probe or reagent kit used for Illumina sequencing. It is common to have read lengths of 36bp, 50bp, 100bp, 150bp or 250bp from Illumina machines. In general, longer read lengths are desired to provide enough information to reconstruct the sequence for various reasons including: ability to map and assemble less ambiguously, span repeats better, haplotype phasing, splicing events, isoform and indel detection and detection of new and clearly resolve structures. In some cases, short read lengths are also desirable. e.g. cases where read counts matter or where coverage depth is desired (DNA mixture quantification)

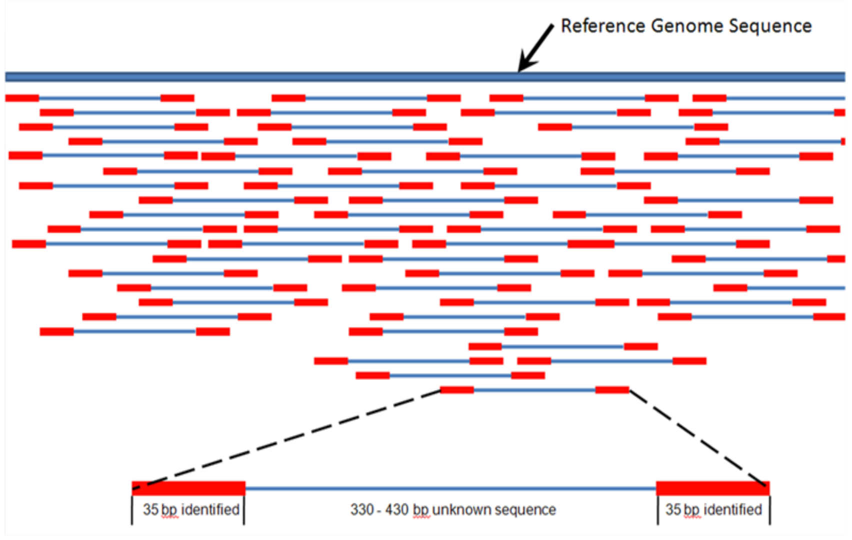


Figure 2.9 The blown up read is 2 x 35. The insert size (bp between the sequencing adapters) is 400-500 bp. All the reads in the picture could be used to assemble a single contig based on the consensus sequence.

2.5.3 Ambiguous or Uncalled Bases

Ambiguous bases (also known as uncalled bases) are usually represented by N's in the data. These N's can be any of A, C, T and G nucleotide. To infer the ordering of nucleotide bases during a sequencing reaction, a process known as base calling is initiated. Base calling is performed using base calling software programs. The accuracy of the Phred base-calling software makes it one of the most widely used software [51]. It works by identifying the unique colors of the reversible dye terminators associated with each base in a next generation sequencing platform. The program accuracy in inferring or calling a base during a sequencing run is typically measured by a quality score known as

Phred quality score and is related to the base calling error probability by Q = - 10 log P / log 10. Despite the software accuracy, it sometimes cannot determine the definite value of some bases. Ambiguous nucleotides are used when the true nucleotide is unknown resulting in ambiguous or uncalled bases in the sequence data.

2.5.4 Sequence Duplication

Duplicate reads in sequence data are introduced during PCR library amplification process [6]. Copying the subsequence of a DNA multiple times as a DNA replication process during PCR steps creates these duplicate reads. Since every read in the sequenced data is expected to come from a different fragment of the original experimental sample during analysis, duplication may constitute a bias and affect correct reporting of analysis result. These duplications can be procedural, systematic, or true sample duplicates. Procedural duplication will exaggerate the number of observations. Repeated observations will cause any count based analysis to have unwarranted amounts of confidence in measures coming from the duplicated data. Systematic bias can be identified when duplication is not equally applicable to each fragment but preferring one fragment over another. Low duplication levels are indicative of high unique reads while the opposite indicates high duplicate reads. Having too many reads for a small target and enrichment bias are the two most likely causes of duplicate reads. Reduced duplication levels can usually be achieved by reducing the number of required PCR cycles.

2.6 Errors in Next-Generation Sequencing

Since its discovery, genome sequencing has become the foundation of virtually every area of molecular biology and new genome research. Large- scale high-throughput genome sequencing has rapidly emerged resulting in generation of an astronomical amount of genomic data in a manner faster than Moore’s law. Figure 2.10

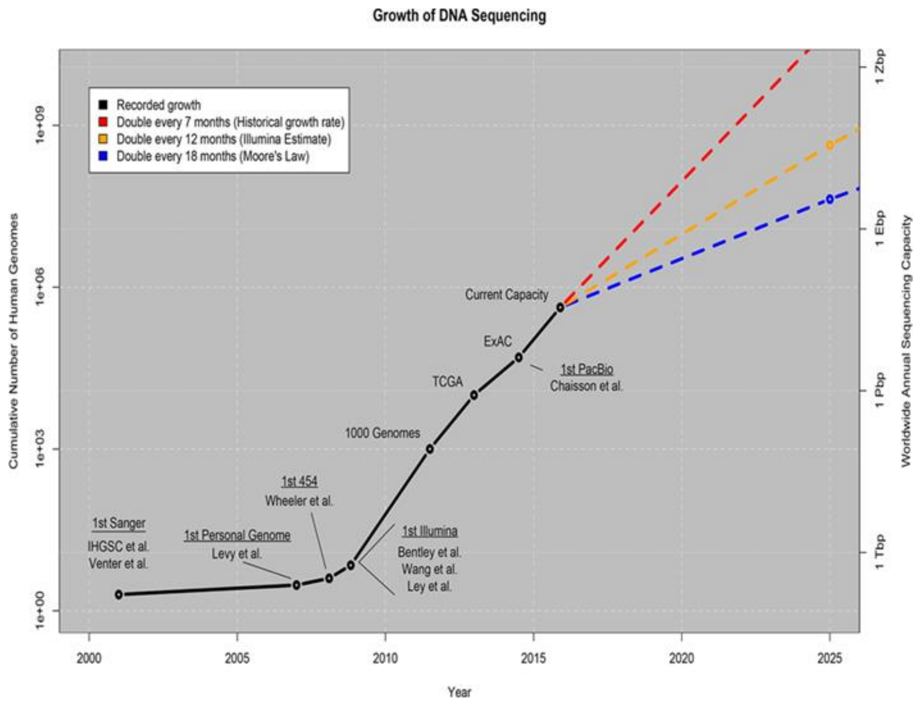


Figure 2.10 Growth of DNA sequencing. The plot shows the growth of DNA sequencing both in the total number of human genomes sequenced (left axis) as well as the worldwide annual sequencing capacity (right axis: Tera-base pairs (Tbp), Peta-base pairs (Pbp), Exa-base pairs (Ebp), Zetta-base pairs (Zbps)). The values through 2015 are based on the historical publication record, with selected milestones in sequencing (first Sanger through first PacBio human genome published) as well as three exemplar projects using large-scale sequencing: the 1000 Genomes Project, aggregating hundreds of human genomes by 2012; The Cancer Genome Atlas (TCGA), aggregating over several thousand tumor/normal genome pairs; and the Exome Aggregation Consortium (ExAC), aggregating over 60,000 human exomes. The values beyond 2015 represent projections. Figure and corresponding text adapted from [63]

The growth in the data size is continually expected but there is a need to generate useful data. The advancement in technology together with the ability of digitalizing DNA have so far aided sequencing of several highly accurate genomes. However, NGS data have been known to be error prone allowing its usefulness to be called to question if no further processing is performed to remove or correct the errors.

Sequencing errors also known as mis-call occurs when a sequencer incorrectly calls one or more bases. Since DNA sequencing is laboratory based and with molecular biology been so peculiar, there is no guarantee of precisely calling a base during sequencing. These biases and errors are introduced at different stages of the sequencing experiment thereby impacting downstream data analysis. Despite the availability of numerous NGS error correction methods, certain drawbacks like high false positive rates, resource consumption and high time complexity of the methods and their algorithms remains a menace.

2.6.1 Types and Characteristics of NGS Errors

Various factors determine the errors that can creep up during NGS sequencing. The factors can be because of the machine itself (systemic), knowledge level of the individual performing the sequencing (experimental) or Biological which is often due to the actual DNA being sequenced. For example, presence of oil on an Illumina GAII slide, over or under sampling the DNA, use of expired reagents and DNA properties like homopolymer can all constitute error sources during sequencing. Clone pre-processing, gel length used in the electrophoresis process and sequencing primers [52] [53] also causes a variation in the data quality observed over each sequence strand.

Sequencing platform selection for given experiment is also critical to error reduction and requires in-depth knowledge of a machine's error sources and rates. This is because different sequencing platforms have their own benefits and shortcomings. For example, the distribution of error types varies from one platform to another [6].

Furthermore, different sources contribute to noise in the sequencing reaction Figure 2.11, while at the same time, a relationship exists between the noise source and the error type that results. Despite the emergence of newer sequencing technologies, the trust worthiness of the data generated is still not 100% accurate as several unnoticed artefacts can be a source of sequence quality degradation. Characterizing the errors generated by common NGS platforms and differentiating true genetic variants from technological artefacts are unrelated steps, essential to various downstream analysis like variant calling, haplotype inference, genome assembly and evolutionary studies.

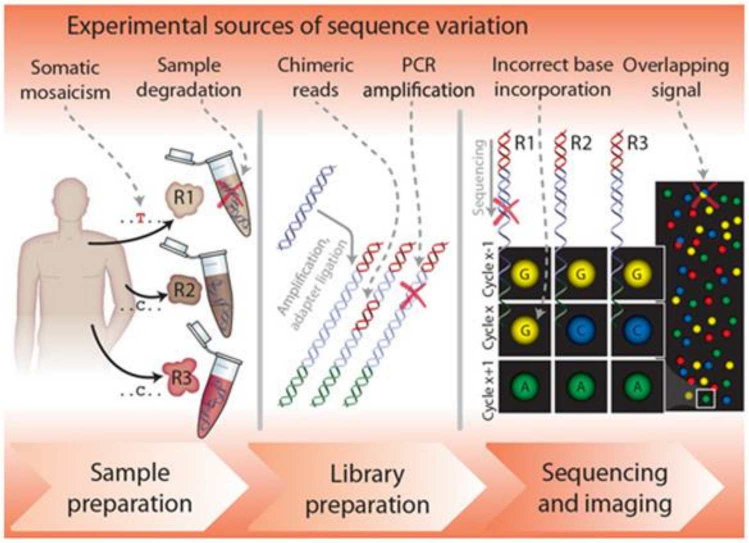


Figure 2.11 Sources of noise during sequencing reactions adapted from [54]

In general, NGS technologies contains higher amount of errors than Sanger sequencing while SMS technologies has a much higher error distribution than that of PCR-based methods. Errors occurring in NGS data includes insertion, deletion errors together known as indel errors, and Substitution errors. Indel errors are less common than Substitution errors which are the predominant error type in NGS data. As an example, the dominant error type in Illumina and SOLiD reads are mostly substitution errors[55], reads from 454/Roche platform tend to contain many insertions and deletions due to its inability to correctly assess the length of homopolymer runs [56][57]. Meanwhile, Helicos reads contains more of deletion errors (2–7% error rate with one pass; 0.2–1% with two passes).

Substitution error otherwise known as mismatch occurs when a certain nucleotide is replaced by a different nucleotide leading to a wrong base been called. [58] reported that miscalls are more frequent during the first and last cycles and proposed that Illumina-specific miscalls result from cycle-dependent variations of the cross-talk matrix, declining intensities, pre-phasing and phasing and T accumulation. According to [59], miscalls are more frequently distributed in the GC-rich regions. The authors also claimed that the base-specific miscalls A to C and C to G are observed more often than the others, suggesting that this type of miscall is due to the inhibition of base elongation during SBS. Various researchers agree that the quality of the Illumina sequencer reads is significantly lower in the later cycles. Lagging-strand dephasing, caused by the incomplete extension of the template ensemble, has been suggested as one of the main reasons for this problem [60]. In genome sequencing terms, dephasing refers to a situation whereby there is no synchronization of the various amplified DNA fragments due to inaccurate de-blocking and incorporation of nucleotide sequences on the DNA strand.

Errors that result from misjudging the length of homopolymer runs leads to single-base insertions and deletions (indels). Insertions occur when a wrongly called base is placed where there is no base while deletion is when a base is totally not called in the sequence. The ‘plus and minus’ method of Sanger and Coulson faced the same problems. Nonetheless, the second generation 454 Genome Sequencer FLX is reportedly able to produce 100 Mb of sequence with 99.5% accuracy for individual reads averaging read over 250 bases in length. Insertions and deletions of bases (indels) that produce frame shifts in deduced coding regions cause errors in predicted protein sequences and compromise the interpretation of the chromosome sequence.

A reduction in all the above-mentioned errors can be achieved by using advanced algorithms for base calling, improving the sequencing chemistry or using algorithms for image processing as described in [61] [62] and [63] respectively

2.6.2 Error Rates of Sequencing Platforms

Without doubt, the biggest problem with sequencing analysis is the error content. Measuring the error rate of different sequencing platform is challenging making a direct comparison very tough to accomplish. This is because of the difference in library preparation methods, error calculation algorithms, sample to be sequenced and variation in the read length produced by these platforms. We can only come up with error distributions because the data will not provide any pronounced distinction among the sequencers.

Numerous attempts have been made to identify, enumerate and comprehend errors that are generated by NGS platforms. Errors patterns linked to Illumina platforms have been studied by [59] [64] while those for 454 Genome Sequencers have been studied by [56]. These studies have been instrumental in understanding the NGS quality characteristics. Table 2.1 gives a recent comparison of the error distribution

Table 2.1 Information based on company sources alone from early 2012 (independent data not yet available as of Feb 2014); it is not clear if the 4% error rate reported by Oxford Nanopore refers to a single-pass rate or is what is achieved after reading both strands & producing a consensus sequence; nor is it clear what the error rate will be for instruments when they are released.

INSTRUMENT	PRIMARY ERRORS	SINGLE-PASS ERROR RATE (%)	FINAL ERROR RATE (%)
3730XL (CAPILLARY)	substitution	0.1-1	0.1-1
454 ALL MODELS	indel	1	1
ILLUMINA ALL MODELS	substitution	~0.1	~0.1
ION TORRENT – ALL CHIPS	Indel	~1	~1
SOLID – 5500XL	A-T bias	~5	≤0.1
OXFORD NANOPORE	deletions	≥4*	4*
PACBIO RS	Indel	~13	≤1

2.6.3 Error Detection and Correction

Output generated by NGS sequencing platforms contain short reads that represent the sequence sample. The presence of errors as discussed earlier causes a significant difference from the true genomic sequence. Detecting the errors is the most important step before the actual error correction. Therefore, there is a need for a good procedure to detect the errors in the sequence.

Several error detection methods have been developed in recent years. Most of these procedures cater to errors in genomic sequences. They mostly use the sequence coverage statistics to identify the erroneous base/bases that exists in the sequence data. Coverage statistics is being used because errors occur in sequences in a random manner hence the sequence itself is random. This implies that the generated sequence is not the same as the genomic sample that was sequenced. Therefore, it makes it more probable for those random bases introduced to be seen only a few times (less than thrice). Seeing a rare sequence in a high coverage sample most often indicates that the sequence is erroneous [65] [66].

Often, a threshold is chosen and sequences with multiplicity below the selected threshold gets eliminated from the read. To do this, the coverage must be uniform (which follows a Poisson distribution) or high throughout the whole genome sequence. We must however note that there is a distinction between under sampled but true sequence and erroneous sequence with high coverage. A rough estimate of the expected coverage of a sequence was shown in 2.5.1

A careful analysis based on the method described above is required because removal of true sequences which are like other low abundance or rare erroneous sequences may occur. DNA sequence anonymities like repeats or low coverage makes it difficult for errors to be detected. Other methods to improve sequencing error detection have been developed using PCR, Circle Sequencing and coding theory as described in [67][68] and [69] respectively.

2.6.4 Error Correction

As explained above, NGS technologies offer great new opportunities for biomedical sciences, but the data generated is not flawless. Sequencing errors make processing of the data more difficult. DNA sequence assembly struggles with such sequencing errors. Error correction involves identification and use of non-error containing overlapping reads in a sequence to resolve sequencing errors. It is the task of analyzing the data and removing sequencing errors, so that the data can be analyzed more effectively. Error correction is only performed on the raw sequence data therefore, does not require a reference genome. This is a demanding task from an analytical perspective as well as from its computational complexity.

2.6.5 Error Correction Algorithms

Over 61 error correction methods are known and have been evaluated. This includes those built as stand-alone tools and those that have been incorporated into other tools like genome assemblers with the error correction process being just one part of their process. Most importantly, the fundamental structure behind their correction process remains the same. They organize the reads in data structures so that they can be compared with each other to identify the errors and subsequently correct them.

In general, error correction algorithms for correcting genomic sequence data execute three important steps:

- Computation of overlaps among the read sequence
- Detection of the errors in the read
- Read error correction

Mostly, the steps mentioned above gets accomplished based on three main assumptions:

- An incorrectly called or erroneous base per position will be rare in comparison to the correct base in the sequence read
- There should be uniformity of coverage across a sequence data eligible for error correction
- The probability of introducing errors (substitution and indels) is similar at all positions of the sequence sample.

However, error rate is data dependent making these expectations precarious. Also, the rate is basically unknown because of system biases that influence the read coverage and frequencies with which these errors occur. This makes it imperative for more refined methodologies to be developed for error correction. Algorithms by [70][71][72] or [73] approach the error-correction problem with varying strategies. There is a measurable positive effect on mapping or assembling NGS data after first applying an error correction algorithm. Strategies employed for error correction in current methods are further discussed in the next section.

2.6.6 Classification of Error Correction Algorithms

Refinement in error correction algorithms over the years have led to increase in the number of categories of error correction algorithms. Currently, they are classified into five categories with each having many variations in application by different tools. An extensive review have been made in [74][75][76]. Differences between each category and a brief explanation will be addressed with more focus on k-spectrum based methods.

2.6.6.1 K-spectrum Based

K-spectrum based error correction algorithms has the broadest set of applications. It has its origin from deBruijn based assemblers [66][72]. It involves the use of k-mer frequencies for the correction process. K-mers refer to all the possible subsequences (of length k) from a read obtained through DNA Sequencing. This algorithm basically depends on the application of a coverage threshold, determined from a k-mer coverage histogram Figure 2.12 to predict if a k-mer is part of the actual genomic sequence.

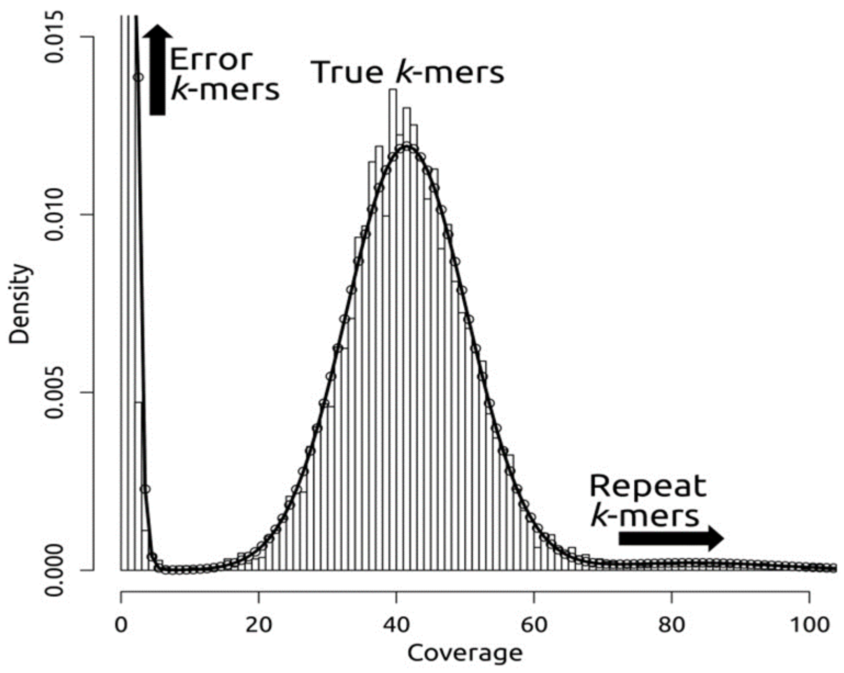


Figure 2.12 k-mer coverage histogram with a model fit. The histogram in this plot from the Quake paper [72] gives a nice example of an empirical k-mer coverage distribution. The density tells us which proportion of all existing k-mers in the data set has a coverage. The solid line gives the Quake model fit. The first peak of the distribution is formed by very low coverage error k-mers and is usually modelled by a Poisson or a Gamma distribution. The second peak results from most correct k-mers and is usually modelled by a Poisson or a Gaussian distribution. Between these two peaks, a clear local minimum can provide a k-mer trust coverage cut-off. The heavy tail of higher multiplicity k-mers is the result of k-mers from sequence repeats. Adapted by font change and label addition from [72]

The original reasoning behind this method is to generate a spectrum of k-mers, set a threshold and consider k-mers above that threshold as most likely part of the sequence while those below the threshold are grouped as candidates for error correction. These are then systematically edited into high-multiplicity k-mers., one full read sequence is inspected at a time. If the k-mer spectrum of the read contains k-mers not in the trusted spectrum of the full sequenced sample, the read is either corrected by the minimum number of base edits necessary to turn all its untrusted k-mers into trusted ones or discarded if that is not possible without making more than a pre-defined maximum number of changes to the read in question.

Several variations of this K-SPECTRUM approach exist. Some can use mixed models to select the required parameters [72][77]. Others decompose reads into k-mer sets [66], [78]–[82]. Yet, some consider the quality values of the bases in their corrections to be better able to discriminate between low copy true k-mers and high copy error k-mers [70]. some assemblers use variants of this approach as pre-processing steps in their assemblers [83]–[86]. The simplified version of how k-mer spectrum is used by some methods is shown in Figure 2.13

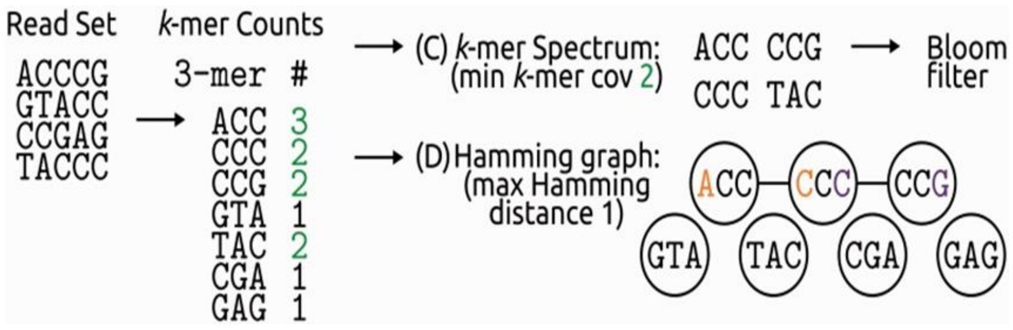


Figure 2.13 Deriving a k-mer Spectrum or a Hamming graph from k-mer counts. Some error correction tools work directly with the k-mer frequencies as counted from the read set. Others set a minimum k-mer coverage (2 in this example, green) to consider a k-mer as correct (trusted k-mers, green counts) and then derive a (C) k-mer Spectrum of all trusted k-mers. See [87]

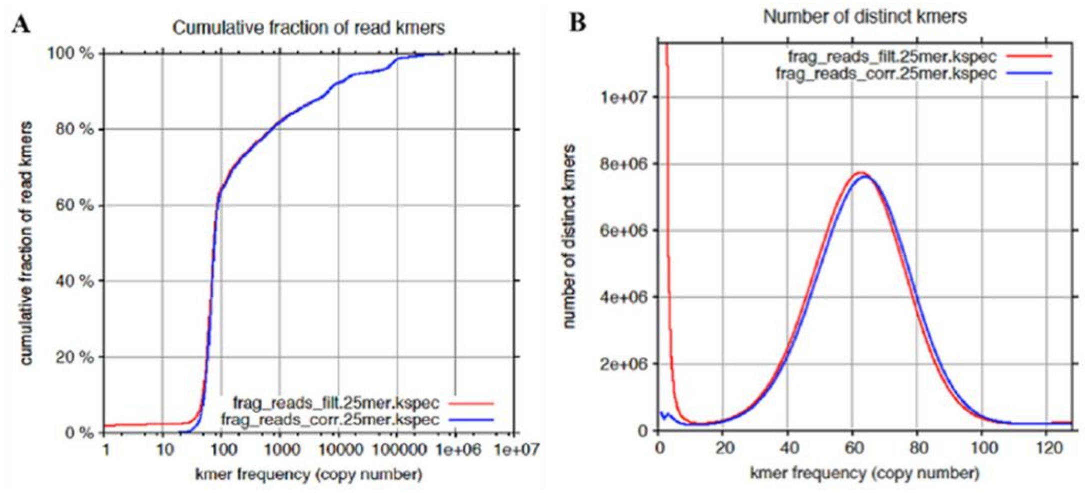


Figure 2.14 depicts the cumulative fraction of all k-mers in the reads as a function of frequency. The spectrum for the filtered reads starts off at about 10%. This means that 10% of all the k-mers in all the reads have very low frequencies and are most likely associated with errors. However, remember that a single base error in a read spoils K k-mers, so the base error rate is not 10%. The base error rate is more like 10% / K. The corrected cumulative spectrum (blue) starts a 0%, as expected for a mostly error free data set. Adapted from [87]

2.6.6.2 Suffix tree/array Based

The generalization of this method to handle multiple kmer values at once is the main difference between the suffix tree-based method and the k-spectrum based approach. Like the k-spectrum algorithm, multiple variations of this approach exist. Basically, the first ever implementation of this method [88] built a suffix tree of all the reads and transverses this tree to find and correct erroneous reads using the k-mer frequency weights associated with the suffix tree nodes.

The reads used for the correction and the erroneous reads are children of branching nodes on the tree with k−1 string depth. Variations of this approach includes: those that use simple extensions like the hamming distance to correct insertion and deletion errors and color space sequences [89] methods that combine statistical approach with partial suffix array, methods that do not use the suffix array but based on the same concept [73] and those that uses coverage statistics to automatically select and set parameters for the error correction [90]. See 2.3.1 for more discussions on suffix arrays.

The problems with this approach is that some of the methods show disparity in performance due to their sensitive nature to input parameters. For others, a large amount of memory is required for the array data structure. For some, automatic selection of parameters occurs only for reads of equal length. Most worrying of all, no explanations are given for how correction is done when the sequence contains the same error occurring multiple times.

2.6.6.3 Multiple sequence alignment (MSA) Based

As the name implies, this approach is alignment based. Alignment identifies similarities between two sequences. Error correction is performed based on how the reads in the sequence aligns with a given reference. Multiple alignment has been used for a long time [91], [92] were used to correct sanger sequences. The general idea is that each individual read is considered a reference. Multiple alignment is performed to determine reads that share a minimum of one k-mer with that reference. The first read is set as the initial consensus of the alignment. The correction of the reads is then based on these alignments and their consensus. Aligning the reads one after the other against the consensus using a variant of the Needleman–Wunsch algorithm [93] allows error correction to be made.

As with other approaches, several variants of this method are available [94] creates a consensus sequence after each alignment with the reference. [95] uses a maximization algorithm that performs pairwise alignment among reads sharing at least one k-mer. [96]uses a directed acyclic graph instead of a consensus sequence. [81] groups read based on spectral clustering before applying MSA. The main problem associated with this approach is that alignment is computationally expensive for long reads.

2.6.6.4 Hidden Markov model (HMM) Based

Basically, HMM is a machine learning probability model whereby a system is modelled under the assumption that it is a Markov process with latent states. It provides a foundation to label and model the states of sequence by the transition probabilities between the states. Used for sequential or temporal data and played a big role in the human genome project.

Sequence reads can be modelled as a de Bruijn graph whereby traversing the graph can be a read and the graph is an HMM (Hidden Markov Model). The vertices (k-mers) on the graph represents the states while the transformation matrix depends on the edges. A recursive transversion of an edge gives off one base in a read whose identification maybe be prevented by sequencing errors or may not be present. Since repeats constitute only a part of the genome, selecting a reasonable k is possible. This way, the subsequent character strongly depends on only k characters before it, forming the basis of the Markov assumption. For error correction, the problem is posed as a maximum likelihood sequence detection problem [97], [98]. The problem with this approach is that a large state space is required. This is because small k will increase speed but as reads gets longer, the computation becomes more complex.

2.6.6.5 Hybrid Based Methods

Due to different characteristics of the sequencing platforms, it is an attractive idea to combine reads produced by several platforms. Most hybrid error correction methods are seemingly meant to correct long or third generation sequences (TGS) reads. It is based on combining two sets of reads with one set as the reference and the other as the sequence candidate for error correction. The idea is to marry the long reads like the error prone long single molecule reads from PacBIO with the highly abundant and relative high quality but short read length of NGS like Illumina Miseq

Downstream analysis can be affected by the coverage of TGS data no matter the read quality due to the uneven distribution of the read length. Since TGS data mostly have lower coverage due to the sequencing cost, the higher coverage from second generation sequences can be combined with the TGS data to correct the sequence. It is believed that this approach will correct the reads regardless of the coverage of the TGS data. It also prevents computational complexities that may arise because of pairwise alignment between the long reads of TGS data.

Variations of the hybrid approach includes: those that use de Bruijn graphs [99] and those that perform multiple alignment between the two read sets in order to call the consensus sequence [100]–[102]. They are all mostly targeted towards PacBIO reads.

2.7 Data structures for Error Correction

Having briefly walked through classifications of error correcting algorithm in 2.2, we are aware of how error correction is handled by most of the methods. The most important take away is that all methods involve the use of k-mer subsequence in one way or the other. They all look for a way of differentiating correct k-mers from erroneous ones to create a consensus sequence. K-mer counting is one of the methods most widely used to determine the frequency of k-mer occurrence. Once a k-mer is selected, it is important to efficiently process it for the error correction to be made e.g. a need to identify and store the observation of each k-mer. Data structures are employed for this purpose

Data structures allow the structuring of sequence data (DNA strings) in a time and memory efficient manner. The efficient handling of these strings is necessary for almost all error correction techniques that are applied to bioinformatics sequences. In this section, we briefly explain the data structures that have been previously employed in bioinformatics but place more emphasis on the newer data structure which forms the basis of our work.

2.7.1 Suffix Arrays

A suffix array, a space-efficient data structure, contains a sorted array of all possible read suffixes. Figure 2.15. It basically acts as a means of storage for suffix trees. The construction of arrays and trees are similar except that practically suffix arrays are preferred for storing or presenting a suffix tree. Suffix arrays supplemented with additional tables can substitute for suffix trees [103]

Suffix array SA(T) = an array giving the lexicographic order of the suffixes of T. It is often used in conjunction with an array termed LCP array, containing the lengths of the longest common prefixes between every consecutive pair of suffixes in SA.

Practically, using an array has the advantage of space efficiency (lower memory consumption) but its performance over using full trees may be sacrificed depending on how it is used. Theoretically, it is very simple to construct. Suffix arrays can be built easily in O (n * log2n) time, where n is the length of the sequence. Using linear time sorting algorithm, it can be built in O(nlogn) time and a search for a pattern of length m can be done in O(mlogn) time by a binary search; reduced to O(m+logn) as in the case of using LCP.

The popularity of suffix arrays in bioinformatics is evident from their application in a range of tasks such as pairwise sequence alignment [104]–[107], error correction of reads from high-throughput sequencers [88], [90], prefix–suffix match finding for genome assembly [108], [109], k-mer counting [110] and sequence clustering [111], as well as the development of suffix array software explicitly aimed at bioinformatics applications [112].

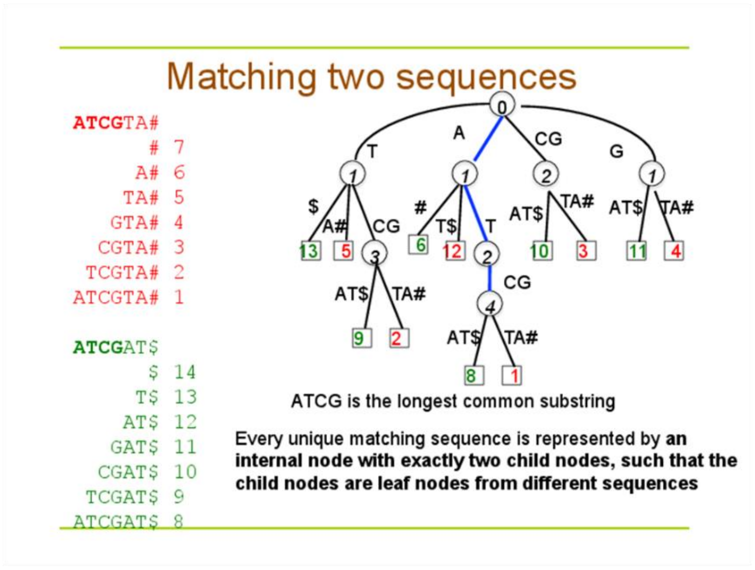

Matching two sequences
ATCGTA#
# 7
A# 6
TA# 5
GTA# 4
CGTA# 3
TCGTA# 2
ATCGTA# 1
ATCGAT$
$ 14
T$ 13
AT$ 12
GAT$ 11
CGAT$ 10
TCGAT$ 9
ATCGAT$ 8
ATCG is the longest common substring
Every unique matching sequence is represented by an internal node with exactly two child nodes, such that the child nodes are leaf nodes from different sequences

Figure 2.15 A suffix tree and array matching two strings adapted from [113]

2.7.2 Hash Tables

A hash table is a data structure used to map keys (indices) to values of an array as a pair. To put it simply they are used to store a set of sequence data for efficient search within the set [114]. As opposed to arrays which uses integers as indices, a hash table can use a string, floating point values or an array itself as the index. A hash table is made up of two parts: an array (the actual table where the data to be searched is stored) and a mapping function, known as a hash function. The hash function provides a way for assigning numbers to the input data such that the data can then be stored at the array index corresponding to the assigned number. i.e. it maps a block of i bits to a smaller block of j bits. Two properties that should be matched by a hash function include;

- The mapping must be balanced: same number of combinations of the i bits maps to each one of the combinations of the j bits.
- Similar values of the input map to different values of output

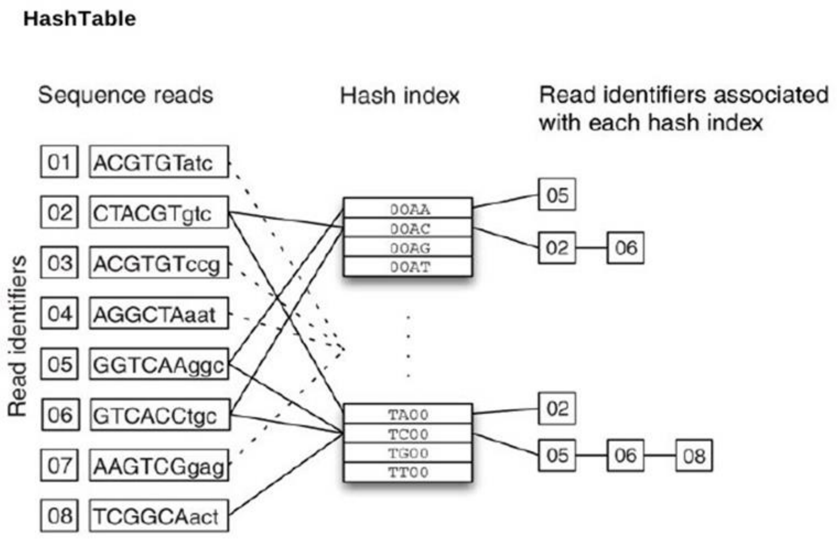


Figure 2.16 Hash tables example.

Hash tables are commonly stored in memory and can consume a significant amount of space. The data is therefore exposed to the errors that affect the memory such as radiation induced soft errors [115]. To avoid data corruption in memories, ECC (Error Correction Code) is commonly used [116]. ECCs add additional bits to each memory word which are used to detect and correct errors and therefore increase the area and power consumption of the memory.

Error correction techniques for next-generation sequencing, such as RACER [89], employ the use of hash tables to achieve efficient storage of k -mers. The basic idea is to hash the read sequences into a hash table and scan through the reference sequence with the same sequence lengths as the original data to find the exact matches in the hash table of reads. Hash tables face certain problems like; effective determination of the hash table size, problems of collision resolution, storage in memory and resource consumption.

2.7.3 Bloom Filters

Originally developed by Burton H Bloom [115], a BF (Bloom Filter) is a simple probabilistic data structure used to test set membership with high space efficiency and fast look-up times. It allows querying of a set to see if an item is not in the data set using just a few bits. In other words, it is not used to test if an element is present but to test whether it is certainly not existing in the set.

BF's do not give any false negatives and the risk of generating false positives (incorrectly presenting an element as a member of the set) is manageable with a trade-off of memory and time. Its compact representation is the payoff for allowing a small rate of *false positives* in membership queries. It is basically a one-sided error data structure because of it guarantees no false positives. Because of this guarantee, no extra work is done to search for elements that do not exist in the set. Figure 2.17 shows an example of a BF with three (3) hash functions.

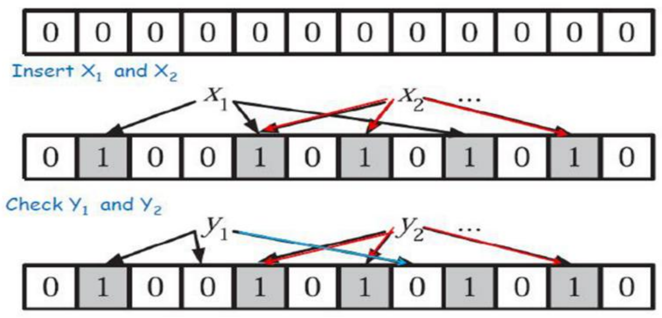


Figure 2.17 Bloom filter example with three hash functions adapted from (Simon S Lam)

BF performs multiple hash of an object by either using multiple hash functions or one hash function with different seed thereby ensuring that same outcome is unlikely to be achieved when an object is hashed. The multiple hash functions are used to minimize false positives. The hope is that between all the k-hash functions, each value will have a unique signature in the bit-array compared to every other possible value. For each time, an object is hashed, the equivalent hash value in the bit array is marked as 1. This is the reason for using a bit array. Instead of requiring 4 bytes to store a 1 or a 0, a bit can be used. Using this technique, elements are hashed independent of their size. When a search is then performed, each hash function is used and checked to make sure their bit-values are all 1s. It is important to control the false positive rate. To do this, the size of the data set has to be known or chosen. [116] presents the methods behind choosing the values. Upon insertion of *n* keys into a filter of size *m* using *k* hash functions, the probability that a given bit is still 0 is:

$$p_0 = \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-\frac{kn}{m}}. \tag{1}$$

Therefore, the probability of a false positive (the probability that all *k* bits have been previously set) is:

$$p_{err} = (1 - p_0)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \tag{2}$$

In (2) p_{err} is minimized for $k = \frac{m}{n} \ln 2$ hash functions. The number of hash functions used, practically, is small because the overhead remains constant as more hash functions gets added but the benefits of each addition decreases after a specific limit.

Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees, tries, hash tables, or simple arrays or linked lists of the entries. Most of these require storing at least the data items themselves, which can require anywhere from a small number of bits, for small integers, to an arbitrary number of bits, such as for strings. Bloom filters do not support deletion, thus removing even a single item requires rebuilding the entire filter

Variations of bloom filters exists. Like the counting bloom filter [117] which support delete operations by extending the bit array to a counter array but requires four times more space than a standard Bloom filter and Compressed bloom filter [118] which has a cost of processing time for compression and decompression.

Several error correction methods for NGS data have extensively used variants of bloom filters as their underlying data structures. BLESS [80], [119]–[124], have all made significant improvement by using bloom filters. Despite this improvement, it has been shown that there remain some problems with the false positive rate generated and computational resource consumption in terms of space and time. This is because of the limitations associated with bloom filters and its variants even though bloom filters have been shown to be the best of all the data structures discussed so far. A look at a more advanced data structure is imperative to determine if better or comparable results can be achieved upon implementation.

2.7.4 Cuckoo Filters

The main reason for the extensive overview of BF (bloom filters) in 2.7.3 is because of its similarity to CF (Cuckoo Filter). CF can replace BF for approximate set membership tests for error correction methods. One major limitation of Bloom filters is that the existing items cannot be removed without rebuilding the entire filter in addition to the false positive rate. Several proposals have extended classic Bloom filters to support deletion, but with significant space overhead: counting Bloom filters [125] are 4× larger and the recent DL-CBF (d-left counting Bloom filters) [117], which adopt a hash table-based approach, are still about 2× larger than a space-optimized Bloom filter. As a practical data structure, its four main advantages are:

- It supports dynamic addition and removal of items
- It achieves higher lookup performance
- It is easier to implement than alternatives such as the quotient filter
- It requires less space than a space-optimized Bloom filter when the target false positive rate ε is less than 3%.

Basically, it is meant for applications that store many items and while targeting moderately low false positive rates. Cuckoo filter [126] is a compact variant of a cuckoo hash table [127]. It does not store fingerprints of the entire items but only for each item that is inserted in the table. Cuckoo hash tables can have more than 95% occupancy, which translates into high space efficiency when used for set membership.

A cuckoo filter uses a hash table to store a small fingerprint for each element, and answers queries by testing whether the fingerprint of the queried element is present. Each element has two hash table cells where its fingerprints might be stored, determined by a combination of a hash of the element and a second hash of the fingerprint. As in cuckoo hashing [127], fingerprints already stored in the table may be moved to their second location to make room for a newly inserted fingerprint. The performance of a cuckoo filter is controlled by the number n of elements in the set it represents, together with three design parameters: the table size N (number of cells), block size b (fingerprints that can be stored in a single cell), and fingerprint size f (bits per fingerprint). A good choice of these parameters allows the fingerprints for all elements in the given set to be stored in the table, giving a data structure whose false positive rate (the probability that an element not in the set is falsely reported to be in the set) can easily be bounded by $\leq 2b/(2f - 1)$. For bad choices of parameters, or unlucky choices of hash function, the data structure may fail, being unable to store all its elements' fingerprints. Therefore, it is important to analyze the likelihood of a failure, and to understand which combinations of parameters have a guaranteed low failure probability

2.8 Kmer Based Error Correction Methods

Because errors located at precise genomic position occurs rarely and in a random manner, reads that cover that genomic position will occur with very high frequency. Kmer based methods take advantage of this high frequency to detect and correct any base considered erroneous. All kmer-based error correction methods employ this idea by counting *K*-mers and storing the counts using various data structures. For example, BLESS [119]employs a bloom-filter and RACER [89] organizes 2-bit-encoded *K*-mers as 64-bit integers and stores them in a hash table.

2.9 Previous Related Work

Musket [120]uses a multi-stage workflow including two-sided conservative correction, one-sided aggressive correction, and voting-based refinement. It computes the multiplicity of each k-mer in the hash table to filter out the stored unique k-mers using Bloom filter. A parallelized slave-master k-mer counting method is implemented to sort out unique k-mers and then generates k-mer coverage histograms to determine a cut-off for a k-mer spectrum for the coverage of likely correct and erroneous k-mers. The error correction stage initially uses a two-sided conservative correction strategy to correct one substitution error, at most, in any k-mer of a read with the intention of finding a unique alternative base that makes all k-mers covering the position trusted. Significant improvement in speed can be achieved by evaluating only the leftmost and rightmost k-mers that cover the position. It then applies a one-sided correction to aggressively correct errors in the case of more than one error occurring in a single k-mer. Furthermore, to confine the number of false positives, error correction is conducted for each integer value from 1 to the maximal allowable number of corrections. The drawback is its reliance on alternative selection if a k-mer is wrongly called to be trusted even though it contains sequencing errors or incorrect corrections. To overcome this drawback, look-ahead validation and voting-based refinement are implemented to assess the trustiness of neighboring k-mers that cover the base position at which a sequencing error likely occurs.

BLESS [119] uses a single minimum-sized Bloom filter and disk-based k-mer counting algorithm like DSK (Disk Streaming of K-mers) [128] and KMC (K-mer Counter) [129] to achieve high memory efficiency for error correction, sequence repeat handling, and read end correction by read extension. Briefly, it counts k-mer multiplicity to sort out solid k-mers from weak k-mers, creates a k-mer multiplicity histogram to determine the multiplicity threshold M, and programs those solid k-mers into a Bloom filter. Weak k-mers are converted to their canonical forms using consecutive solid k-mers (known as k-mer islands) in their neighborhood or read end through Bloom filter querying. Bases in a weak k-mer that do not overlap with solid k-mers are modified. For instance, weak k-mers that exist between two consecutive solid k-mer islands S 1 and S 2 are corrected by using the rightmost k-mer of S 1 and the leftmost k-mer of S 2. BLESS has three distinctive features: high memory efficiency, better handling of genome repeats, and more accurate error correction at read ends.

As part of the GATB (Genome Assembly & Analysis Took Box) [122], Bloocoo was developed to correct large datasets with low memory footprints by using DSK [33], a counting algorithm that requires a user to define a fixed amount of memory and disk space. Its error correction process is like CUSHAW [130], a procedure also used by Musket. In Bloocoo, the multi-set of all k-mers present in the reads is partitioned, and partitions are saved to disk. Then, each partition is separately loaded into memory in a temporary hash table. The k-mer counts are returned by traversing each hash table. Low-abundance k-mers are optionally filtered and solid k-mers are inserted in the Bloom filter based on a given threshold. With a multi-stage correction approach like Musket [120], correction is performed by scanning k-mers of a read, trying the other three different possible nucleotides at the error site, and checking if corresponding k-mers are in the set of solid k-mers. When several close errors occur, the pattern is more complex, and errors are corrected via a voting algorithm. Bloocoo distinguishes itself from other error correctors in the k-mer counting stage and the way that solid k-mers are stored in memory. By using only 11 bits of memory per solid k-mers, Bloocoo requires only 4-GB memory for the entire human genome re-sequencing read correction at 70× coverage.

Different from other tools, Lighter [124] samples k-mers randomly, i.e., sub-sampling fraction α rather than counting k-mers. It uses a pattern-blocked Bloom filter [131]to decrease the overall number of cache misses and improve memory efficiency. Lighter populates Bloom Filter A with a k-mer subsample, followed by a simple test applied to each position of each read to compile a set of solid k-mers, and then stores the solid k-mers in Bloom Filter B. A sequenced k-mer survives sub-sampling with probability of α, a user determined sub-sampling fraction that is set to be 0.10 (70/C) with

C being average coverage. For error correction, lighter applies a greedy approach like that used in BLESS [119] and extends a read when an error is located near the end of the read. Error correction is parallelized by using concurrent threads to handle subsets of the reads. Lighter maintains near constant accuracy and Bloom filter size if the sampling fraction is adjusted in inverse proportion to the coverage depth. However, a user should specify k-mer length, genome length, and sub-sampling fraction α.

Trowel [121] is a highly parallelized and efficient error correction module for Illumina sequencing reads. The key difference to other tools is that Trowel relies on contiguity of high quality values instead of a k-mer coverage distribution to differentiate between solid and weak k-mers. The algorithm not only improves low quality bases but also iteratively expands the trusted k-mer set by including corrected k-mers. Trowel applies two different algorithms, DBG (Double Bricks & Gap) and SBE (Single Brick & Edges), to increase the likelihood that a correction can be made and to boost quality values. The DBG algorithm exploits an asymmetric k 1-gap-k 2 structure, where a gap is a single base, $k = k\ 1 + k\ 2$. The quality of the gap is boosted to the maximum quality value when the index relevant to gap-enclosing bricks contains the gap with high quality. The SBE algorithm is used because bases at read ends cannot be accessed by the brick index. Hence, a new edge-k-edge index is used to correct edges, where an edge is a single base, or increase their quality values as in the DBG algorithm.

BFC [123] is the most recent error correction method and unlike some of the above algorithm which use greedy approach for correction, it uses a non-greedy method and corrects recurrent errors of low base quality in a read. It is fast and handles compressed data. BFC encounters problems in correcting errors that exist in regions of low coverage. Which is synonymous with k-mer based error correction methods.

2.10 Discussions

For the accuracy of these error correction methods, various factors had to be taken into consideration. One factor is the handling of ambiguous bases in the sequence which are often represented by other characters apart from ACTG. The way these ambiguous bases are handled is critical to their performance. Also of importance are the error correction parameters determined by each individual method. How the parameters are selected and assignment of default values is critical for correction accuracy. Both factors are discussed in this section

2.10.1 Ambiguous Base (N) Handling

A fair and precise analysis of the error correction methods is highly dependent on how each method deals with ambiguous or unknown bases mostly existing as N’s within a sequence read. Different methods handle N’s in different ways. BFC [123] tries to correct N’s to A/C/T/G based on k-mers around it while BLESS remove ambiguous bases by choosing large k-mer values and any base that does not overlap with the solid k-mer gets modified to a base found in the solid k-mer. i.e. Ns are converted to an arbitrary character and corrected if the character does not match with the original ones. Furthermore, Trowel corrects all ambiguous bases if it has a low quality (Phred score < 38) during the Single Brick Edge stage because it finds a candidate base based on high quality anchoring k-mer statistics. On the other hand, lighter corrects N’s but does not attach any importance to the resulting correction in calculating precision. This is because it uses scores for the calculation and scores corrections at bases represented by N a value of zero (0) making it irrelevant. Musket only randomly converts unknown bases to known bases which, we assume, may impact the correction performance. Bloocoo [122] also converts ambiguous bases to known bases before correction.

2.10.2 Parameter Selection

One of the most important aspect of error correction and indeed, any NGS analysis, is how parameters for running the analysis are selected. Constant features of most of the error correction methods are options for both user specified and default parameters. This is to give control to the user because a variety of dataset exist. Selecting what options to turn on or off is critical depending on a user's understanding of their data. There also parameters that should be left untouched because they are automatically determined based on the given dataset or a prior analysis that confirms such parameters to be optimal across board. Developers will almost always provide instructions for parameter selection. If not given, a careful study of the method employed and the available examples of running the error correction algorithm is always a good start for correcting NGS data. Several articles reviewing various error correction methods also provide recommendations on how to select some parameters for reviewed methods.

2.11 Significance of Error Correction

NGS data are used in several analyses and many of these analyses may benefit from error correction. Analysis like identification of variation in copy number, chromosomal rearrangement, genome assembly, SNP (Single Nucleotide Polymorphism) etc. can all gain from error correction.

A known example of the significance of error correction is in genome assembly. Due to the size of both plant and animal genomes, it is currently impossible to sequence the genome at once. This led to sequencing of genomes in fragments. These fragments are aligned and merged to recover the original genome sequence through a process known as genome assembly. It is basically fusing together the ordering of the bases that make up the DNA. The generated reads are assembled into a draft genome by identifying reads originating from the same region. Once identified, they are merged into longer contiguous sequences called contigs. The short length of sequence reads makes assembly complex therefore requiring highly reliable data for accurate assembly. However, the flawed nature of NGS data because of introduced sequencing errors, leads to unrealizable downstream analysis. The introduction of false genetic information complicates such analysis. Error correction allows for recovery of data that could otherwise muddle the work of a genome assembler. Genome assembly has been extensively discussed in [132]

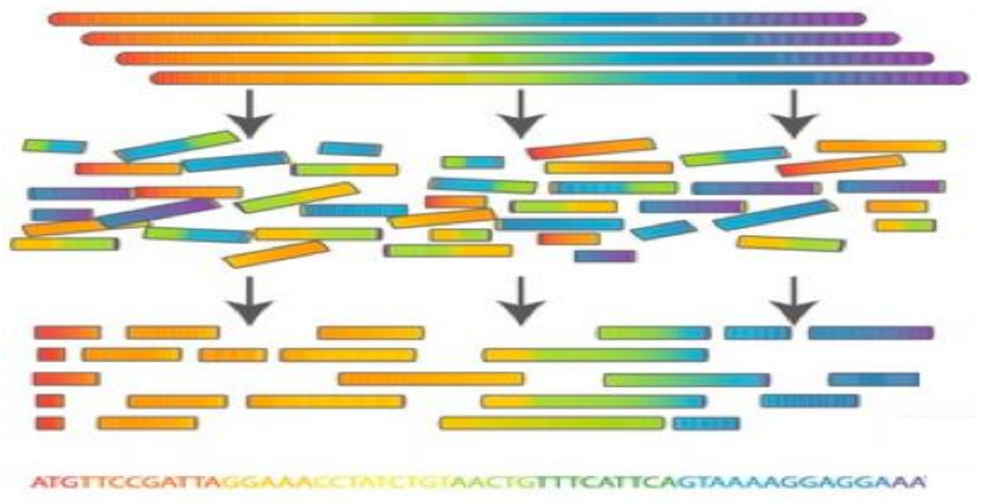


Figure 2.18 Assembly: http://people.mpi-inf.mpg.de/~sven/images/assembly.png

It has also been well studied that error correction affects sequence alignment. Errors in a read sequence can lead to misalignment of reads to a reference genome. For repetitive regions, errors in a unique path of a read can allow it to match multiple locations of the repetitive region [78], [93], [120], [130].

Error correction is also of significance during re-sequencing where a comparison to understand variability between multiple known genomes of a given specie is desired. In this case, the data maybe from same organism with a known reference, sequenced using different sample preparation methods or sequencing platforms [73][133].

Furthermore, errors also affect SNP detection and discovery. To detect variants, sequence aligners usually map the read to a reference genome. Once mapped, these errors can increase the observed differences resulting in a deceptive analysis. Also, SNPs are not uniformly distributed and errors often have high impact on regions of high density [72], [73].

Error correction is not without its problems too. Even though the software is proposed to correct errors, erroneous correction like changing a base from the correct to an incorrect value, can also introduce errors into the sequence. A careful examination is required before correcting the data.

2.12 Motivations for k-spectrum based application

Error correction methods based on k-spectrum originate from earlier implementation of de Bruijn graph assemblers using spectral alignment [66] and follow a generalized framework as shown in Figure 2.18.

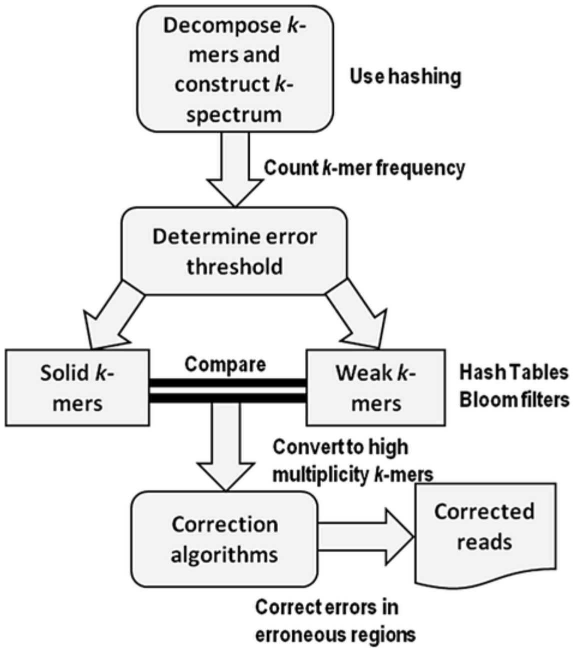

Decompose *k*-mers and construct *k*-spectrum
Use hashing
Count *k*-mer frequency
Determine error threshold
Solid *k*-mers
Compare
Weak *k*-mers
Hash Tables
Bloom filters
Convert to high multiplicity *k*-mers
Correction algorithms
Corrected reads
Correct errors in erroneous regions

Figure 2.19 General framework of k-spectrum based error correctors

A k-spectrum is the distribution of a set of decomposed distinct substring of length k (i.e., k-mer) observed in a group of reads. It counts the occurrence of all k-length contiguous strings represented as a vector within the spectrum feature space. The expectation is that errors in a sequence will result in a strong divergence at low k-mer frequencies compared to a sequence without errors. One challenge in error correction is that inconsistent genome sampling and genomic repeats may occur at high frequencies and consequently result in numerous equally susceptible correction possibilities. Owing to this, a frequently explored property of the k-mer spectra is the distribution composition of the spectra representing motif groups with varying sequence and bias frequencies [134]. This implies that based on their frequencies of occurrences, k-mers having small hamming distances are presumably of the same genomic locus and should be corrected. K-spectrum-based correction starts by assigning a weighted value to each k-mer after extraction from sequencing reads. The value is assigned based on sorted count frequencies or base quality scores.

By determining and selecting an acceptable error threshold [135], [136], weak (insolid or untrusted) k-mers with low frequencies are separated from solid (trusted) k-mers (with high frequencies). The reads with weak k-mers are considered for error correction by repeatedly converting them into solid k-mers until there are no weak k-mers in the sequence. Hence, only solid k-mers will be kept after correction.

2.13 Problem Statement

Despite the availability of over 60 error correction methods of which approximately 59.1% of them are k-mer frequency and spectrum based, there still exists limitations in their correction ability. The amount of false positive corrections generated can still be further reduced by implementing more robust correction methods. There is a need for an in-depth comparative statistical analysis to determine how these correction methods currently perform. The amount of computational resources used is also desired to be minimal which is currently not the case. Though computational resources like storage and RAM (Random Access Memory) are getting cheaper in addition to existence of cloud based resources, there is also a rise in the length and size of NGS data with improved sequencing technologies. Therefore, it is important for error correction to be achieved with high accuracy and minimal resource usage. Furthermore, most error correction methods are also Linux command-line based, making it difficult for molecular biologist, with little to no experience working on Linux systems, to correct their NGS data. There is currently no web-based platform for error correction. A web-based platform that will provide a user-friendly interface for error correction is necessary. All the above-mentioned problems necessitated the need for our current studies.

2.14 Chapter Summary

This chapter discussed different types of error correction algorithms and data structures that were and may be implemented. It outlined the significance of error correction for NGS data analysis and the problems encountered by current correction methods. It details the significance of error correction to genome assembly, sequence alignment, SNP detection and re-sequencing projects. Based on the studies, the chapter defined the problems that will be tackled in the dissertation.

CHAPTER III - COMPARISON OF ERROR CORRECTION METHODS

3.1 Framework

Having discussed different error correction methods in CHAPTER II, it is imperative to analyze the performance of the methods to gain full insight into their error correction process, advantages and disadvantages. The evaluations allow making of informed recommendations to users on what, why and when they should use a given error correction method and how to choose their parameters. The analyses also led to the provision of a web-based workflow called BECOW (Bioinformatic Error Correction Workflow). This workflow makes it easy for Molecular Biologists to correct their NGS data without the need to learn command-line language which may constitute a steep learning curve for them. The analysis was performed in two different phases. Results from phase I evaluations led to a more robust evaluation in phase II. Though the purpose of both evaluation phases is the same i.e. to study performance of the existing error correction methods and make usage recommendations to users, the evaluation methods are different:

- Phase I – involved a direct evaluation of performance metrics.
- Phase II – involved evaluation of performance metrics from a statistical perspective by expanding on phase I.

Both analysis phases are presented in 3.5 and 3.6

3.2 Evaluation Workflow

The workflow used for both phases of the analysis are the same as shown in Figure 3.1 Steps taken in the workflow consists of: sequencing dataset simulation, pre- and post-correction alignment to reference genome, parameter optimization for error correctors, and derivation of evaluation statistics and metrics (see http://aluru-sun.ece.iastate.edu/doku.php?id=ecr for more details). Briefly, both error-free and error-containing paired-end sequences were generated in FASTQ format by ART simulation. The error-free data served for the QA/QC purpose throughout the workflow. After converting FASTQ to FASTA (pre-process due to ECET's header requirements before alignment), simulated erroneous sequences were aligned to a reference genome using BWA package version 0.7.12 [137]. A reference genome file is used for alignment because it can help in the determination of the difference between the corrected and uncorrected data file after error correction. The SAM alignment files produced by BWA were then converted to TEF format using ECET (Error Correction Evaluation Toolkit) [75]. ECET version 1.1 was used to allow an unbiased comparison. Error-containing datasets were corrected using error correction tools. The error correction outputs from these tools were converted to TEF (Target Error Format) files in ECET. The two TEF files that were generated pre- and post-correction were compared using the Comp2PCAlign script provided in ECET. This generates files that inferred performance assessment metrics of the error correction methods. The metrics are:

- True positive (TP) – a wrong base correctly changed to its true base
- False positive (FP) – a correct base incorrectly altered
- False negative (FN) – an incorrect base left unaltered

- True negative (TN) – a true base correctly identified
- Recall or sensitivity = TP/ (TP + FN)
- Precision = TP/ (TP + FP), gain = (TP − FP)/ (TP + FN), and F-score = 2 × ((precision × recall)/ (precision + recall)).

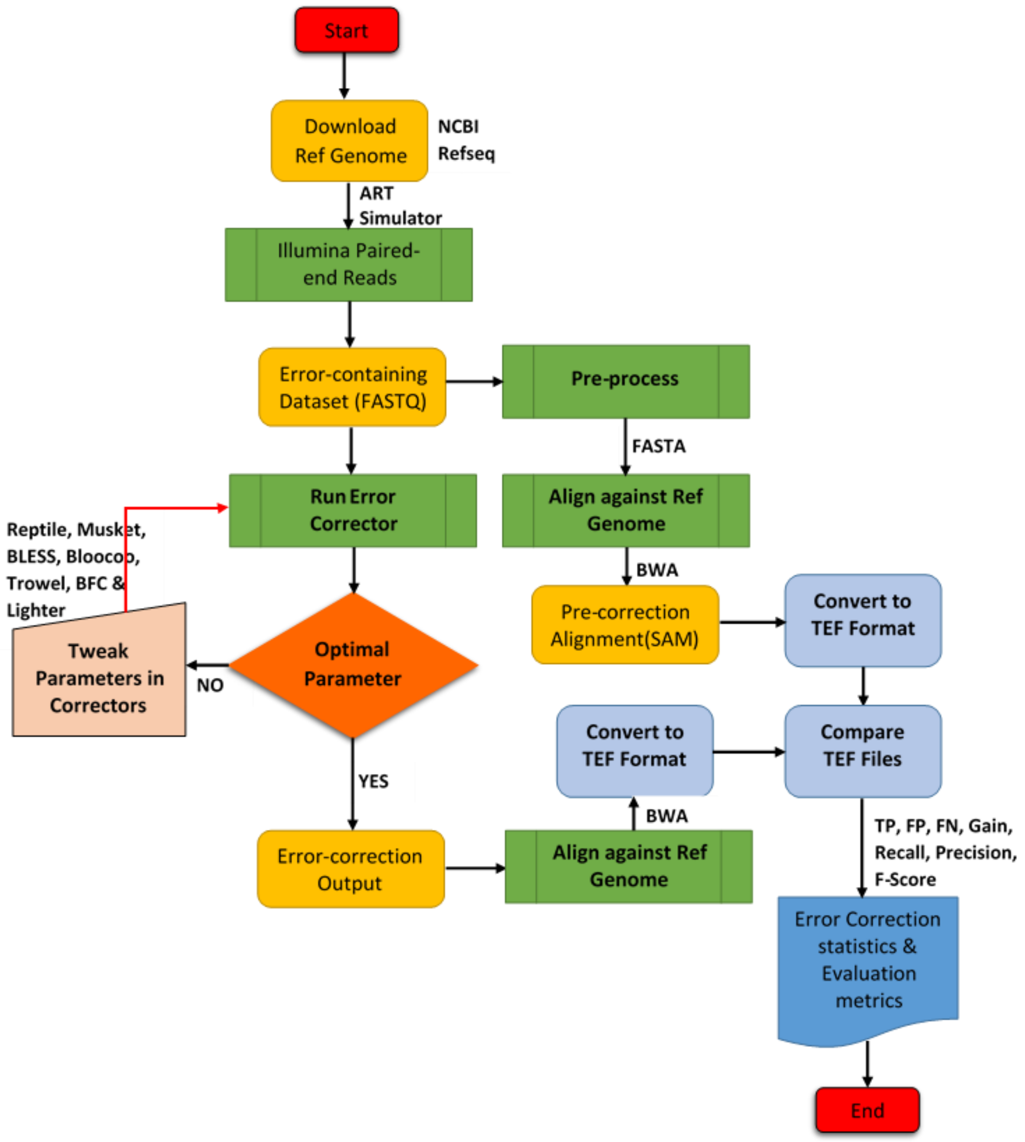

Start
Download Ref Genome
NCBI Refseq
ART Simulator
Illumina Paired-end Reads
Error-containing Dataset (FASTQ)
Pre-process
FASTA
Run Error Corrector
Align against Ref Genome
Reptile, Musket, BLESS, Bloocoo, Trowel, BFC & Lighter
BWA
Tweak Parameters in Correctors
NO
Optimal Parameter
Pre-correction Alignment(SAM)
Convert to TEF Format
Convert to TEF Format
Compare TEF Files
YES
BWA
TP, FP, FN, Gain, Recall, Precision, F-Score
Error-correction Output
Align against Ref Genome
Error Correction statistics & Evaluation metrics
End

Figure 3.1 Workflow for error-correction performance analysis using ECET. See [75] for more information.

The generated metrics have been widely used to evaluate error correction quality [75], [80], [119]–[124] Despite the importance and uniqueness of each of the metrics generated, the focus of our analysis was on precision and F-score. F-score, defined as a harmonic mean of precision and recall, was chosen because it accounted for both false positives and false negatives which should have similar cost to downstream data analysis (e.g., genome assembly). Precision was chosen because it provides the accuracy of corrections made by a given correction method based on the percentage of right corrections within the total corrections for an individual dataset.

3.3 Evaluation Parameter Setting

An inherent difficulty in using any corrector is the challenge of choosing optimal parameters [80]. Since the quality and accuracy of error correction tools are highly dependent on parameter (particularly k-mer) settings, we introduced an iterative optimization loop to select recommended k values by using Kmergenie version 1.6476 [138]. Very few tools have implemented automated choice of parameters sensitive to datasets being processed. Although BLESS [119] can automatically choose an appropriate value for M, k-mer multiplicity threshold, it cannot select an optimal k and nor can other tools evaluated in this study (except for Reptile [80], which chooses $k = \log4|G|$, where G is the genome length). In this loop, we also tweaked other parameters while maintaining the same suggested best k-mer derived for a specific dataset. For instance, α is a user-defined parameter in Lighter and its default value is set by the formula 0.1(70/C), where C is the coverage depth.

While it is possible that the k picked by Kmergenie may not be the optimal value for all six evaluated tools, we performed limited tests in tweaking k and other user-defined, tool-specific parameters but did not observe significant deviations in terms of performance metrics. For similar reasons, we set edit distance to 2 (36/56-bp reads), 4 (100-bp reads) or 6 (250bp) for read alignment based on the recommendation of 4 % read length (see http://bio-bwa.sourceforge.net/bwa.shtml).

3.4 Selected k-mer based Error Correction Methods

For both phase I and II analysis, 7 recently published and widely used k-mer based error correction methods were selected for the evaluation. i.e. BFC[123], BLESS [119], Bloocoo [122], Lighter [124], musket [120], trowel [121] and Reptile [80]. Reptile was used in phase I but dropped in phase II due to its age (published in 2010) in addition to the complex nature of its execution. Given that it is also single threaded and the large number of dataset that had to be evaluated, it will be unfair to evaluate and compare the speed and computational resource usage of reptile to the other methods. It was replaced by BFC, which was the latest k-mer based method as at the time of writing this dissertation. As summarized in Table 3.1, these methods differ greatly in error correction algorithms as well as in how hash tables and Bloom filters are implemented

Table 3.1 Distinguishing characteristic features of seven k-mer based methods investigated in phase I and II.

Tools	**Algorithm highlight**	**Data structure**	**Pros**	**Cons**	**Quality score**	**Target error type**
Reptile	Explore multiple alternative *k*-mer decompositions and contextual information of neighbouring *k*-mers	Hamming graph	Contextual information to resolve errors without increase in *k* or lowering local coverage.	Uses a single core (non-parallelized)	Used	Substitution Deletion Insertion
BFC	Non-greedy and quality aware method. Uses exhaustive search like in fermi	Blocked bloom filter /hash table	Results in fewer overcorrections; has two variants and handles compressed files	Specifically, for high-coverage WGS human data	Used	Substitution
Musket	Multistage correction: two-sided conservative, one-sided aggressive & voting-based refinement.	Bloom filter	Master-slave Multi-threading results in high parallel scalability	A single coverage cut-off to differentiate trusted and weak *k*-mers	Not used	Substitution
BLESS	Count *k*-mer multiplicity; Correct errors using Bloom filter; Restore false positives.	Bloom filter	High memory efficiency; Handle genome repeats better; Correct read ends	Cannot automatically determine the optimal *k* value	Not used	Substitution Deletion Insertion
Bloocoo	Parallelized multi-stage correction method like Musket	Blocked Bloom filter	Faster and lower memory usage than Musket	Not extensively evaluated	Not used	Substitution
Trowel	Rely on quality values to identify solid *k*-mers; Use two algorithms (DBG and SBE)	Hash table	Correct erroneous bases and boost base qualities.	Only accept FASTQ files as input	Used	Substitution
Lighter	Random sub-fraction sampling; parallelized error correction.	Pattern-blocked Bloom filter	No *k*-mer counting; near constant accuracy and memory usage	A user must genome length and subsampling fraction α	Used	Substitution Deletion Insertion

3.5 Phase I Analysis

3.5.1 Error Correction Methods

In phase 1, six k-spectrum-based methods, i.e., Reptile version 1.1, Musket version 1.1, BLESS v0p23 for 64× Linux, Bloocoo 1.0.4-linux, Lighter version 1.1, and Trowel version 0.1.4.2, were compared using six simulated sets of paired-end Illumina sequencing data.

3.5.2 Computational Environment

Computational experiments were conducted using multiple machines due to specific requirements of individual tools and varied sizes of synthetic testing datasets; hence, consideration was not given to the performance in terms of run time and memory usage but rather effectiveness and accuracy of read error correction. Due to requirement of Message Passing Interface (MPI), BLESS was run on a Red Hat Enterprise Linux MPI cluster with 12 nodes, and each node had 12-GB memory and 8 cores running at a core speed of 2.93 GHz. For all other tools, datasets with a genome size >5 MB were run on a 64-bit Ubuntu 12.04 LTS Intel Core i7-3770 CPU@ 3.40 GHz machine with 8 cores and 8-GB memory. The E. coli datasets (genome size <5 MB) were run on a CentOS—64-bit Intel(R) Xeon(R) CPU E5630@ 2.53 GHz machine with 16 processors and a total memory of 296 GB.

3.5.3 Synthetic NGS Datasets

Reference genome sequences downloaded from NCBI (National Center for Biotechnology Information) RefSeq (Reference Sequence) database, included two bacteria genomes (*Escherichia coli* (EC) strain K-12 and *Bacillus cereus* (BC) strain ATCC 14579); and one invertebrate genome (*Drosophila melanogaster* (DM)). Synthetic, paired-end sequence read datasets were generated using ART with default Illumina profiles of empirical quality score distributions and error rates [139]. We chose to simulate Illumina-specific sequencing data because of the predominant status of Illumina sequencers among all NGS platforms. ART was used because it imitates the sequencing process with built-in, NGS platform-specific read error models and base quality value profiles parameterized empirically using large sequencing datasets [139]. All three types of errors (substitution, insertion, and deletion) for all major sequencing platforms are incorporated in simulated reads. The following default error rates were selected: 0.009 % and 0.015 % of insertion and 0.011 % and 0.023 % of deletion for the first and the second read, respectively. Base substitution is the dominant error type accounting for up to 98 % of all errors in Illumina sequencing data. The substitute rate in the simulated datasets varied, resulting in the overall error rate varying between 0.1 and 0.95 %, which is typical for Illumina sequencers (see Table 2.1). Details of the six simulated datasets are shown in Table 3.1 and these datasets can be downloaded at http://pinfish.cs.usm.edu/ngs_correction/. The simulated NGS datasets varied in coverage depth (10× to 120×), read length (36 to 100 bp), and genome size (4.6 to 143 MB).

Table 3.2 Synthetic paired-end Illumina sequencing datasets simulated using ART.

Organism (dataset ID)	Accession number of reference genome assembly	ART simulation parameter				Genome Size (Mb)
		Read length (bp)	**Genome coverage**	**Insert size**	**Error rate (%)**	
Escherichia coli (EC-1)	GCF_000005845.2 (ASM584v2)	36	70×	200	0.866	4.6
Escherichia coli (EC-2)	GCF_000005845.2 (ASM584v2)	36	20×	200	0.866	4.6
Escherichia coli (EC-3)	GCF_000005845.2 (ASM584v2)	100	20×	200	0.952	4.6
Bacillus cereus (BC-1)	GCF_000007825.1 (ASM782v1)	56	50×	200	0.175	5.4
Bacillus cereus (BC-2)	GCF_000007825.1 (ASM782v1)	100	120×	300	0.109	5.4
Drosophila melanogaster (DM)	GCF_000001215.4 (Release 6)	100	10×	300	0.854	143

3.5.4 Comparative Analysis Results

The derived performance metrics are presented in Table 3.2. BLESS and Bloocoo each failed to process one dataset, i.e., BC-2 and DM, respectively. A negative gain value means that more errors are introduced into the data than corrected. Five methods (Reptile, BLESS, Bloocoo, Trowel, and Lighter) produced negative gains, mostly in processing EC-3. F-score is the most comprehensive measure of error correction performance. If setting F-score = 0.8 as the threshold for good performance, all methods except Musket underperformed with at least one dataset. Therefore, Musket was the best overall performer whereas, for yet undiscovered reasons, Trowel was the worst one with five instances of underperformance.

Table 3.3 Performance analysis of six k-spectrum-based error correctors as evaluated using six synthetic Illumina datasets

Dataset	Method	TP	FP	FN	Recall	Gain	Precision	F-score
EC-1	Reptile	2335361	144751	451889	0.8378	0.7859	0.9416	0.8867
	Lighter	2695425	72843	91825	0.9671	0.9409	0.9737	0.9704
36 bp	BLESS	2624659	48342	***56279***	***0.9790***	***0.9610***	0.9819	***0.9805***
	Bloocoo	2411701	***22259***	375549	0.8653	0.8573	***0.9908***	0.9238
70×	Musket	***2701885***	61096	85365	0.9694	0.9474	0.9779	0.9736
	Trowel	1246340	705438	1539825	0.4473	0.1941	0.6386	0.5261
EC-2	Reptile	681551	140039	114910	0.8557	0.6799	0.8296	0.8424
	Lighter	108241	58579	688220	0.1359	0.0624	0.6488	0.2247
36 bp	BLESS	***779824***	18095	***16637***	***0.9791***	***0.9564***	0.9773	***0.9782***
	Bloocoo	689322	***6454***	107139	0.8655	0.8574	***0.9907***	0.9239
20×	Musket	767087	18182	29374	0.9631	0.9403	0.9768	0.9699
	Trowel	434885	19167	361576	0.5460	0.5220	0.9578	0.6955
EC-3	Reptile	105	461	876053	0.0001	-0.0004	0.1855	0.0002
	Lighter	858125	2446	18033	0.9794	0.9766	0.9972	0.9882
100 bp	BLESS	746	872860	875412	0.0008	-0.9954	0.0009	0.0009
	Bloocoo	79790	3644539	796368	0.0911	-4.0686	0.0214	0.0347
20×	Musket	***873592***	***1645***	***2566***	***0.9971***	***0.9952***	***0.9981***	***0.9976***
	Trowel	155	178354	876003	0.0002	-0.2034	0.0009	0.0003
BC-1	Reptile	382043	22303	16602	0.9584	***0.9024***	0.9448	***0.9515***
	Lighter	331759	***15470***	***141618***	0.7008	0.6682	***0.9554***	0.8086
56 bp	BLESS	***429017***	34018	11943	***0.9729***	0.8958	0.9265	0.9492
	Bloocoo	410156	24127	63221	0.8664	0.8155	0.9444	0.9038
50×	Musket	355015	47460	118362	0.7500	0.6497	0.8821	0.8107
	Trowel	55277	4976	26744	0.6739	0.6133	0.9174	0.7770
BC-2	Reptile	497425	116	208081	0.7051	0.7049	0.9998	0.8269
	Lighter	698089	159	7417	0.9895	0.9893	0.9998	0.9946
100 bp	BLESS	-	-	-	-	-	-	-
	Bloocoo	27409	1278837	678097	0.0389	-1.7738	0.0210	0.0272
120×	Musket	***703882***	***68***	***1624***	***0.9977***	***0.9976***	***0.9999***	***0.9988***
	Trowel	652845	108	52661	0.9254	0.9252	0.9998	0.9612
DM	Reptile	***11702183***	187733	***517322***	***0.9577***	***0.9423***	0.9842	***0.9708***
	Lighter	42	23055867	12224293	0.0000	-1.8861	0.0000	0.0000
100 bp	BLESS	11122683	***126388***	1101652	0.9099	0.8995	***0.9888***	0.9477
	Bloocoo	-	-	-	-	-	-	-
10×	Musket	11550483	163838	673852	0.9449	0.9315	0.9860	0.9650
	Trowel	1197127	384403	11027208	0.0979	0.0665	0.7569	0.1734

In the first column, dataset ID, read length, genome coverage, and the optimal k estimated using KmerGenie are shown. The values in TP, FP, and FN columns are number of bases. Italicized and bolded values denote the best performer given a specific evaluation measure for a dataset. The symbol "–" indicates that a method failed to process a specific dataset

3.5.4.2 Influence of read length on performance

Three datasets with a short-read length of either 36 or 56 bp were processed by four to five methods to a satisfactory degree (F-score > 0.8, Fig. 3.2). Only two, three, and four methods generated satisfactory results with the other three 100-bp datasets EC-3, DM, and BC-2, respectively. In general, read length has an adverse impact on tool performance, i.e., the longer the read length, the less superior a tool performs. This impact was the most pronounced on Bloocoo, which underperformed in all three long-read datasets. Musket was the most resistant tool because it performed well across all six datasets. For the other four tools, there appeared to exist interactive effects among read length, coverage depth and genome size because no clear-cut relationship between read length and performance was observable.

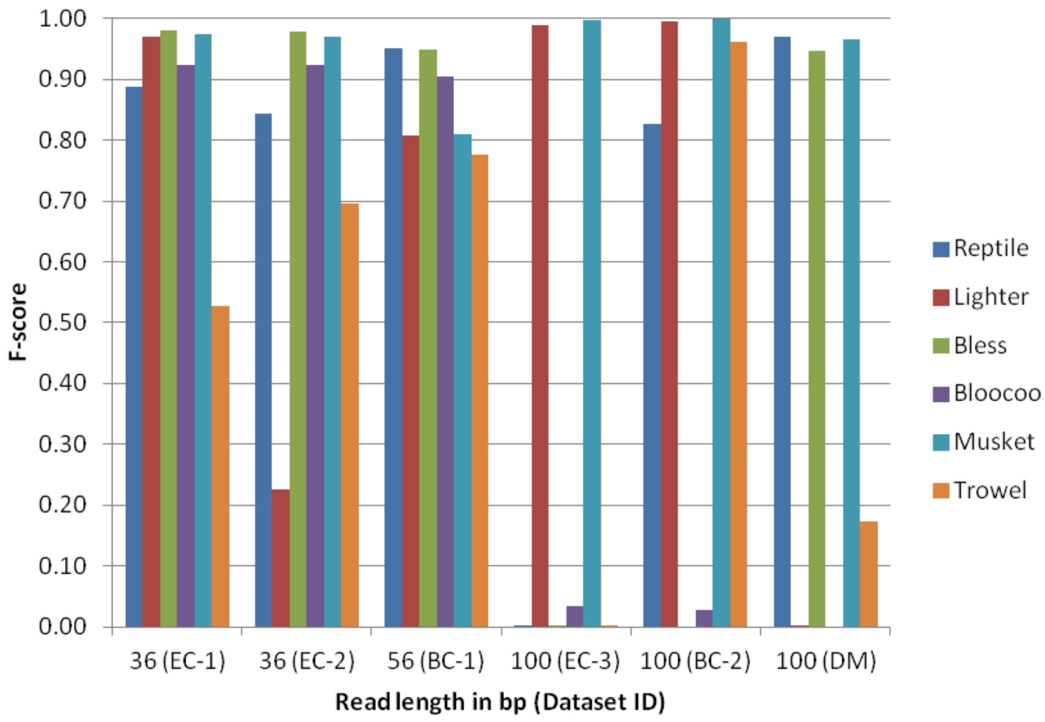


Figure 3.2 Results showing influence of read length on error corrector performance

3.5.4.3 Influence of genome coverage depth on performance

A medium coverage depth (50- and 70-fold) appeared to be preferred by all tested tools except Trowel (Fig. 3.3). At a low depth (10- and 20-fold), Reptile and BLESS performed well except for the long-read dataset EC-3. Lighter seemed to require a medium-to-high coverage depth (50-fold or higher). In case of low depth (20-fold), a longer read length might compensate for the loss of coverage depth, resulting in a satisfactory performance. At the highest coverage depth (120-fold), two tools failed (BLESS) or underperformed (Bloocoo). Again, Musket showed the strongest resistance to variation in coverage depth

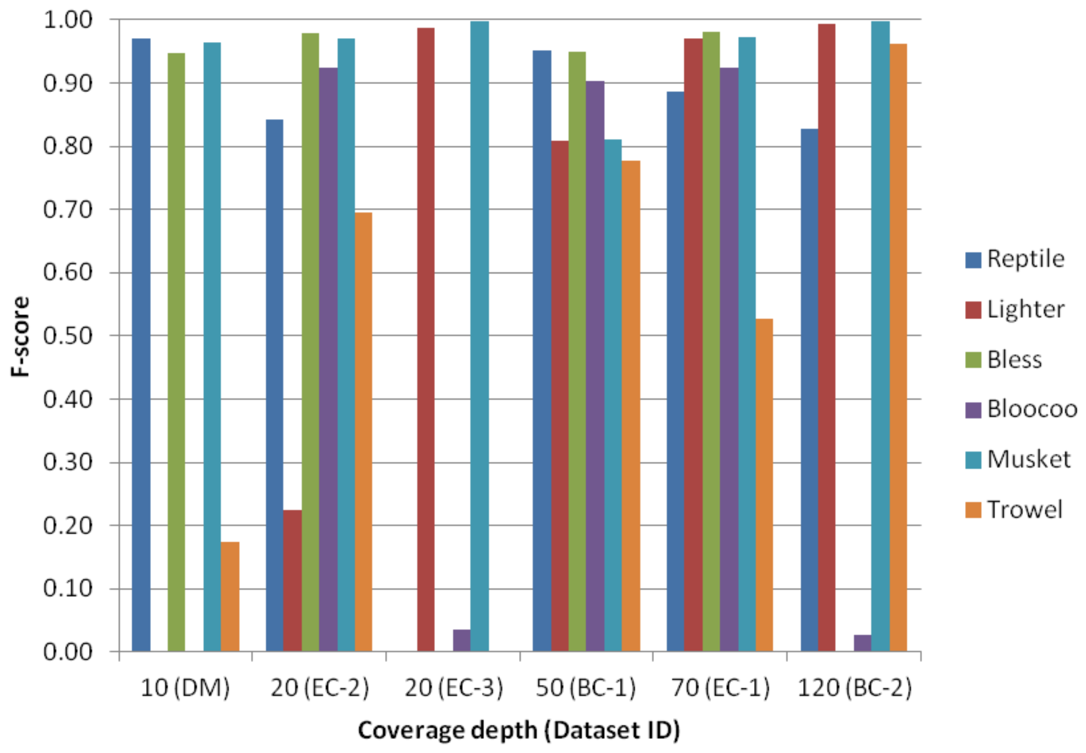


Figure 3.3 Results showing influence of data coverage on error corrector performance

3.5.4.4 Influence of genome size on performance

Genome size is most likely a covariant that interacts with the other two factors (read length and coverage depth) because instances of underperformance occurred across the full spectrum of genome size (Fig. 3.4). For small genomes (EC and BC), Musket was the best method, followed by Lighter and Reptile (both performed well in four of five datasets), then BLESS and Bloocoo (three of five datasets), and Trowel ranked the last. For the large genome (DM), only three methods (Reptile, BLESS and Musket) performed well.

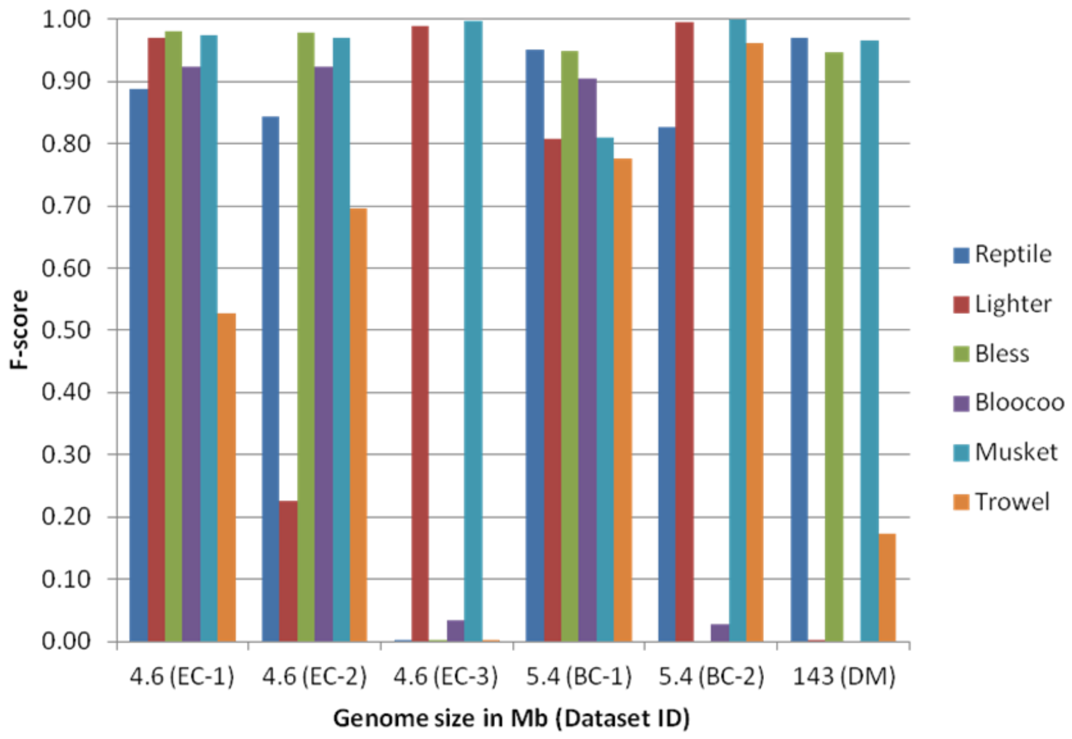


Figure 3.4 Results showing influence of organism genome size on error correction performance

3.5.5 Discussions

Different Bloom filter variants were implemented in four of the six investigated methods to allow compression of the filter, storage of count data, and representation of maps in addition to sets [124] (also see Table 3.1). The other two methods (Reptile and Trowel) used hash tables, which do not yield false positives. Although Bloom filter's space efficiency comes at the cost of false positives, all major error correction programs have reduced or minimized false positive rate by implementing various algorithms. Authors who developed these six tools had put lots of efforts in increasing speed and reducing memory footprint while maintaining or improving their correction quality. In the present study, we chose to focus solely on correction quality because speed and memory are no longer bottlenecking factors that limit the application of these tools.

Simulated datasets were used because correction accuracy could be directly measured. When real experimental datasets are used, only indirect evaluation metrics (e.g., N50 contig size and genome coverage of de novo assemblies and percentage of mapped reads in genome alignment) can be derived for performance assessment. Using real datasets in tool evaluation however, can provide insights that cannot be obtained from simulation studies. Nevertheless, extensive evaluations should be conducted using simulated datasets before moving on to real datasets. Authors of the six tools investigated in our study have performed evaluations using both synthetic and real datasets. In general, tools that perform well with synthetic datasets also work well with real datasets (see publications featuring BLESS [119], Trowel [121], and Lighter [124]). There is a good correlation between performance metrics for simulated and real datasets.

Previous evaluations showed that Musket was consistently one of the top performing correctors for both simulated and real datasets when it was compared with several well-regarded programs: HiTEC, SHREC, Coral, Quake, Reptile, DecGPU, and SGA [84]. Here, we also demonstrated that Musket yielded better performance metrics than Reptile. When authors of BLESS, Trowel and Lighter performed their comparative evaluations, they claimed that their own tools slightly outperformed Musket. However, if looking more specifically into simulated datasets, Musket performed equally well as the other three tools did (e.g., the synthetic 40× human chromosome 1 dataset used in [119]). Bloocoo shares a great deal of similarity with Musket, especially in the multi-stage error correction algorithm [120], [122]. They reportedly achieved similar correction accuracy as measured by recall and precision on a simulated dataset with 1 % error rate from human chromosome 1 at 70× coverage (see "Supplementary Material" in [122]). In the current study, these two programs did perform equally well on three datasets (EC-1, EC-2, and BC-1 with read length of 36 or 56 bp). However, Bloocoo underperformed or failed on the remaining three datasets with longer reads (100 bp), suggesting the existence of potential bottleneck in Bloocoo limiting its application to longer reads.

3.5.6 Review and Future Direction

Identifying and correcting errors in NGS data is an important step before carrying out any in-depth downstream analysis. This phase I analysis was aimed at providing an independent and unbiased evaluation of the effects of three NGS dataset features on the performance of six k-spectrum-based error correction methods with an emphasis on correction accuracy. We observed that performance of six selected methods was dependent on such factors as read length, genome size and coverage depth. Our experimental results suggest that good performance of a method for a specific dataset does not guarantee its ability to perform as well for another type of dataset, hence careful consideration should be given to selecting appropriate tools. Among the six tested methods, Musket appeared to be the front runner, whereas Trowel showed the worst performance. We recommend Musket as the top choice because of its consistently superior performance across all six testing datasets though the datasets and their varieties are limited in scope. In phase II analysis, the analysis was expanded to include the most recent error correction method, a wider spectrum of genome size and complexity (e.g., human genome), and longer reads (e.g., 250bp). Performing a second, more in-depth statistical analysis will give a broader inference on how these error correction methods perform.

3.6 Phase II Analysis

The significance of error correction has previously been outlined. Deficiencies observed in phase I evaluation, see 3.5, of error correction methods includes: limitations due to size of dataset, computational performance and an effective statistical evaluation. Here we report a simulation study using a full factorial analysis to examine how NGS dataset characteristics (genome size, coverage depth and read length particularly) affect error correction performance, as well as to compare performance sensitivity/resistance of six k-mer-spectrum based methods to variations in dataset characteristics. Multi-way ANOVA tests indicate that choice of correction method and dataset characteristics had significant effects on performance metrics (precision and F-score). Overall, BFC, BLESS, Bloocoo and Musket performed better than Lighter and Trowel on 27 synthetic datasets. However, Bloocoo and Lighter were the most resistant to the examined variables, while Musket was the most sensitive method. For each chosen method, read length and coverage depth showed more pronounced impact on performance than genome size. This study shed insights to the performance behavior of error correction methods in response to the common variables one would encounter in real-world NGS datasets. Based on this extensive comparative and statistical evidence of performance, further studies of wet lab-generated experimental NGS data is warranted to validate findings obtained from this simulation study.

3.6.1 Error Correction Methods

In the present study, we selected the following six recently developed k-mer spectrum-based error correction methods: BFC-ht release 181 [123], BLESS version 0.23 [119], Bloocoo version 1.0.5 [122], Lighter version 1.0.7 [124], Musket version 1.1 [120] and Trowel version 0.1.4.3 [121]. All six algorithms represent the state-of-the-art in NGS data error correction and use bloom filter as their underlying data structure. Except BFC, all others were included in our previous study [140]. KmerGenie version 1.6476 [138] was used to determine the recommended best k-mer for each dataset (Table 3.4), which served as the input parameter k during error correction process. All other parameters of the chosen tools were set at the default values

3.6.2 Computational Environment

This simulation study, from data generation and error correction to performance evaluation, was conducted on a dedicated server named BigCat. The server ran on a 64-bit Ubuntu 12.04 LTS Intel(R) Xeon(R) CPU with 16 nodes, 72 GB memory and a core speed of 2.40GHz. The memusage command was used to monitor the amount of memory consumed by each of the correction methods during the error correction process while the time command was used to record CPU time consumption of the process. Both memory usage and time consumption together with the corresponding averages and standard deviations for each corrector are shown in Table 3.5.

3.6.3 Synthetic NGS Dataset

Reference genome sequences of *Escherichia coli* strain K-12 (4.6 Mb), *Drosophila melanogaster* (143 Mb) and Human chromosome 21 (48 Mb) were downloaded from NCBI's reference sequence database (ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq) for NGS data simulation and error correction evaluation. Selection of these three organisms was based on genome size variation, genome complexity (simple prokaryote to complex eukaryote) and taxonomic coverage (bacterium, invertebrate and vertebrate). Ambiguous regions in the reference genomes were removed prior to the simulation study. This was to allow a fair comparison because some methods attach no importance to these regions (e.g. [124]), modifies them ([141], [142]) or randomly convert the ambiguous bases to known bases [120] in their error correction process.

ART, was still used to generate 27 synthetic datasets (9 per organism) of paired-end sequencing reads according to Illumina’s empirical quality score distributions and error rate profile [139]. The simulated reads were 50-, 150- or 250-bp in length with a 20-, 80- or 320-fold genome-wide coverage for each organism. The insert size was kept constant at 300 bp for all simulations to eliminate any bias that may be introduced if variable fragment sizes were allowed. Substitution, deletion and insertion errors were evenly introduced into the sequence at a total error rate of 1%, a rate at the upper limit of reported error rates for most NGS platforms and higher than that of the Illumina platform. This high error rate was chosen to consider worst case scenarios where suboptimal data may be produced from a sequencing run. These simulated datasets are available at http://pinfish.cs.usm.edu/simulation_data.

Table 3.4 Simulated datasets and optimal k-mer values derived using KmerGenie [138]

Organism	**Genome coverage**	**Read length (bp)**	***k*-mer size**
Escherichia coli	20X	50	15
	20X	150	23
	20X	250	35
	80X	50	31
	80X	150	37
	80X	250	59
	320X	50	31
	320X	150	61
	320X	250	76
Drosophila melanogaster	20X	50	18
	20X	150	21
	20X	250	30
	80X	50	15
	80X	150	65
	80X	250	134
	320X	50	36
	320X	150	97
	320X	250	127
Human Chromosome 21	20X	50	17
	20X	150	30
	20X	250	59
	80X	50	31
	80X	150	64
	80X	250	87
	320X	50	37
	320X	150	85
	320X	250	103

Table 3.5 Memory (in GB) and CPU Time (in hours) consumption by six error correctors for the 27 synthetic NGS datasets

Error Corrector	**BFC**		**BLESS**		**Bloocoo**		**Lighter**		**Musket**		**Trowel**	
Measurement	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time
Mean	7.52	5.2	0.02	12.33	4.71	6.29	1.47	3.22	15.55	23.99	18.71	2.42
Standard deviation	4.79	10.6	0	26.8	1.76	15.27	0.56	8.01	19.26	47.17	19.66	4.59
Eco_20X_L50	2.14	0.01	0.02	0.01	4.11	0.01	0.35	0	0.47	0.04	0	0.47
Chr21_20X_L50	2.15	0.02	0.02	0.02	4.12	0.03	0.35	0.02	0.55	0.11	0.01	0.55
Dme_20X_L50	2.16	0.03	0.02	0.01	4.22	0.02	0.65	0.09	1.09	0.16	0.02	1.09
Eco_20X_L150	2.4	0.02	0.02	0.02	4.29	0.14	0.65	0.39	2.83	0.13	0.02	2.83
Chr21_20X_L150	2.87	0.04	0.02	0.01	7.44	0.02	1.27	0.01	2.15	0.26	0.02	2.15
Dme_20X_L150	2.78	0.1	0.02	0.02	4.33	0.02	1.27	0.01	1.43	0.36	0.02	1.43
Eco_20X_L250	3.46	0.11	0.02	0.08	4.36	0.12	1.27	0.05	5.14	0.46	0.04	5.14
Chr21_20X_L250	6.81	0.13	0.02	0.07	4.34	0.09	1.27	0.04	7.5	1.64	0.05	7.5
Dme_20X_L250	2.96	0.08	0.02	0.62	4.43	0.08	1.27	0.05	3.79	1.5	0.08	3.79
Eco_80X_L50	6.36	0.16	0.02	0.11	1.19	0.42	0.84	0.02	1.49	1.76	0.07	1.49
Chr21_80X_L50	3.19	0.06	0.02	0.4	1.19	0.3	0.84	0.02	4.43	0.84	0.18	4.43
Dme_80X_L50	3.99	0.22	0.02	0.33	4.36	0.02	1.47	0.02	2.95	0.78	0.03	2.95
Eco_80X_L150	4.02	0.65	0.02	2.99	4.41	0.51	1.47	0.13	10.45	6.34	0.26	10.45
Chr21_80X_L150	4.51	2.12	0.02	3.54	4.45	0.5	1.47	1.11	15.13	6.36	0.12	15.13
Dme_80X_L150	7.65	2.48	0.02	3.36	4.45	0.98	1.47	1.26	7.69	6.55	0.19	7.69
Eco_80X_L250	7.22	3.71	0.02	8.48	4.49	1.21	1.9	2.91	7.01	10.91	1.89	7.01
Chr21_80X_L250	10.46	5.96	0.02	7.69	4.01	0.84	1.9	2.86	17.31	13.86	3.7	17.31
Dme_80X_L250	9.61	6.94	0.02	9.9	5.01	1.48	1.9	3.18	27.64	35.18	4.17	27.64
Eco_320X_L50	11.93	2.8	0.02	2.79	3.51	3.64	1.9	1.1	12.83	17.82	0.35	12.83
Chr21_320X_L50	12.76	0.25	0.02	1.9	4.49	1.85	1.9	0.81	9.8	17.62	0.95	9.8

Error Corrector	**BFC**		**BLESS**		**Bloocoo**		**Lighter**		**Musket**		**Trowel**	
Measurement	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time
Mean	7.52	5.2	0.02	12.33	4.71	6.29	1.47	3.22	15.55	23.99	18.71	2.42
Dme_320X_L50	8.71	0.22	0.02	1.48	4.49	1.21	1.9	0.84	7.01	10.91	1.35	7.01
Eco_320X_L150	14.96	25.02	0.02	42.69	4.01	2.4	1.9	2.59	17.31	13.86	1.75	17.31
Chr21_320X_L150	15.94	1.61	0.02	7	5.01	14.81	1.9	3.06	57.64	75.18	0.93	57.64
Dme_320X_L150	10.27	1.55	0.02	112	5.09	4.91	1.89	0.25	35.46	212.66	7.66	35.46
Eco_320X_L250	15.94	14.64	0.02	19.92	8.24	22.21	2.22	5.42	66.94	20.78	11.62	66.94
Chr21_320X_L250	13.84	27.41	0.02	22.95	8.25	49.03	2.22	26.44	61.83	72.77	18.39	61.83
Dme_320X_L250	14	44	0.02	84.67	8.98	62.92	2.22	34.22	32	118.99	11.49	32

Eco = E. coli; Chr21 = Human chromosome 21; Dme = D. melanogaster; 20X, 80X and 320X = 20-, 80- and 320-fold coverage; L50, L150 and L300 = read length of 50, 150 and 300 bp

3.6.4 Results of Multivariate Statistical Analysis of Performance Metrics

Multi-way full factorial analyses of performance metrics were generated using SAS (Statistical Analysis System) software, version [9.3] of the SAS System for [Windows] [143] to analyze relationships between two dependent variables (DVs, i.e., precision and F-score) and four independent variables (IVs, i.e., correction method, genome size, coverage depth and read length), as well interaction effects among IVs on DVs. Type III sums of squares were obtained to check the individual effects of the factors, given other factors were included in the model. After that, performance metrics data were stratified into six groups by correction method, and a three-way ANOVA was conducted for each group to examine how the other three IVs affected DVs.

3.6.4.1 Descriptive Statistics of Performance

Two performance metrics (precision and F-score) of the six selected correctors on the 27 synthetic datasets derived using ECET are presented in Figure 3.5. A pictorial view of the TP, FP and FN is also shown in Figure B.1. Among the 162 observations, the 25%, 50% (median) and 75% quantiles are 0.570, 0.852 and 0.967 for precision or 0.291, 0.650 and 0.932 for F-score, respectively. Both metrics display a skewed distribution (normality tests: Kolmogorov-Smirnov $p < 0.01$) with continuous values varying between 0 and 1. In general, ANOVA with fixed factors was modest to the moderate non-normal assumption violation [144], which raised no flag to subsequent factorial analyses. To confirm the conformability of our data to ANOVA test, we performed Levene’s test of variance equality using a means comparison in IBM SPSS Version 24.0. Results shown in Table 3.6 suggest that read length (p-values = 0.058 for F-score and 0.902 for precision), genome coverage (0.424 for F-score) and method (0.221 for F-score and 0.464 for precision) conformed to equality of variance. Although genome coverage (0.008 for precision) and genome size (0.004 for F-score and 0.000 for precision) data exhibited unequal variance, these violations are considered minimal and tolerable. ANOVA test is robust enough to handle such minimal violations. After the full factorial analyses, the adequacy of the normality assumption for each fitted model was checked by residual analysis.

Table 3.6 Result of Levene's test where H_0 and H_1 signifies accepted and rejected hypothesis respectively at a 95% confidence level for hypothesis testing.

Variable	**F Score**					**Precision**				
	DFn	*DFd*	*F Value*	*Pr(>F)*	*Inference*	*DFn*	*DFd*	*F Value*	*Pr(>F)*	*Inference*
Length	2	159	2.893	0.058	H_0	2	159	0.103	0.902	H_0
Genome	2	159	5.778	0.004*	H_a	2	159	41.908	0.000	H_a
Coverage	2	159	0.863	0.424	H_0	2	159	4.941	0.008	H_a
Method	5	156	1.418	0.221	H_0	5	156	0.929	0.464	H_0

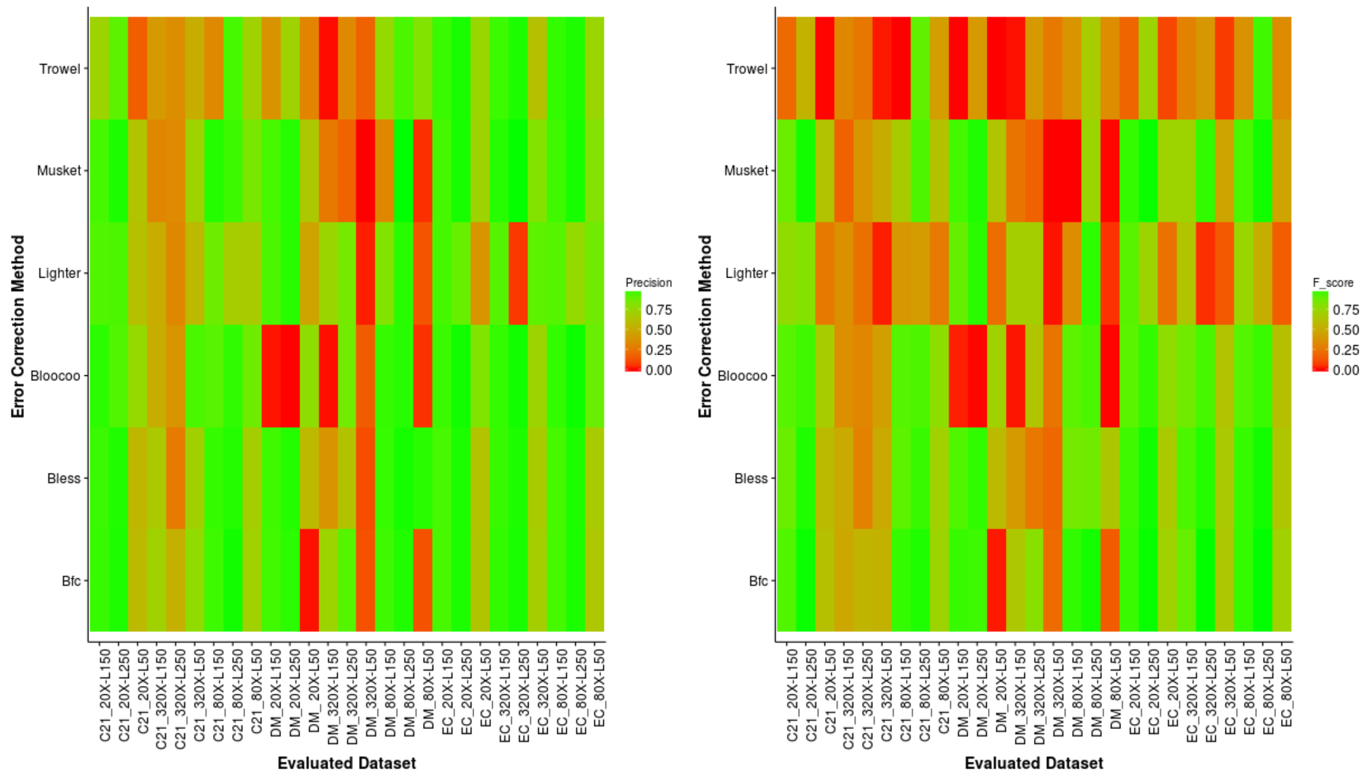


Figure 3.5 Heat maps of F-score and precision for each evaluated method and dataset (see material 2 at http://pinfish.cs.usm.edu/simulation_data/results for their numerical values). C21 = Chromosome 21, DM = *Drosophila melanogaster* and EC = *Escherichia coli*

3.6.4.2 Four-way ANOVA Model

The four IVs were examined for their individual and interaction effects on performance metrics. Significant impact of all four IVs was observed on F-score and precision, except that no significant difference in precision exists between correction methods (p = 0.1206; Table 3.7 and Figure 3.6). The significance of interaction effects differs greatly between the two metrics. Only the pair-wise interactions between method and the other three IVs were observable for F-score, whereas all pair-wise interactions between the four IVs, except coverage × method (p = 0.3320) and length × method (p = 0.0735), were significant for precision (Table 3.8 and Figure 3.7). In addition, a significant 3-factor interaction effect (genome × coverage × length) was also observed for precision.

Table 3.7 Statistical significance expressed as F-test probability for the main and interaction effects of four independent variables on performance metrics (precision and F-score) determined using a four-way ANOVA model. P = p-value, df = degree of freedom.

Dependent Variable	Significant Factor	*df*	Type III Sums of Squares	F statistic	*p*
F-score	Genome (G)	2	1.3546	17.63	<.0001
	Coverage (C)	2	1.9849	25.83	<.0001
	Length (L)	2	3.4061	44.32	<.0001
	Method (M)	5	3.8528	20.05	<.0001
	G × M	10	0.9954	2.59	0.0070
	C × M	10	1.0405	2.71	0.0050
	L × M	10	0.9988	2.60	0.0069
Precision	Genome (G)	2	2.3468	30.34	<.0001
	Coverage (C)	2	1.7245	22.30	<.0001
	Length (L)	2	2.0184	26.10	<.0001
	Method (M)	5	0.3471	1.80	**0.1206**
	G × C	4	0.6234	4.03	0.0045
	G × L	4	0.8551	5.53	0.0005
	G × M	10	0.8831	2.28	0.0187
	C × L	4	0.3816	2.47	0.0497
	C × M	10	0.4456	1.15	**0.3320**
	L × M	10	0.6892	1.78	**0.0735**
	G × C × L	8	0.8724	2.82	0.0073

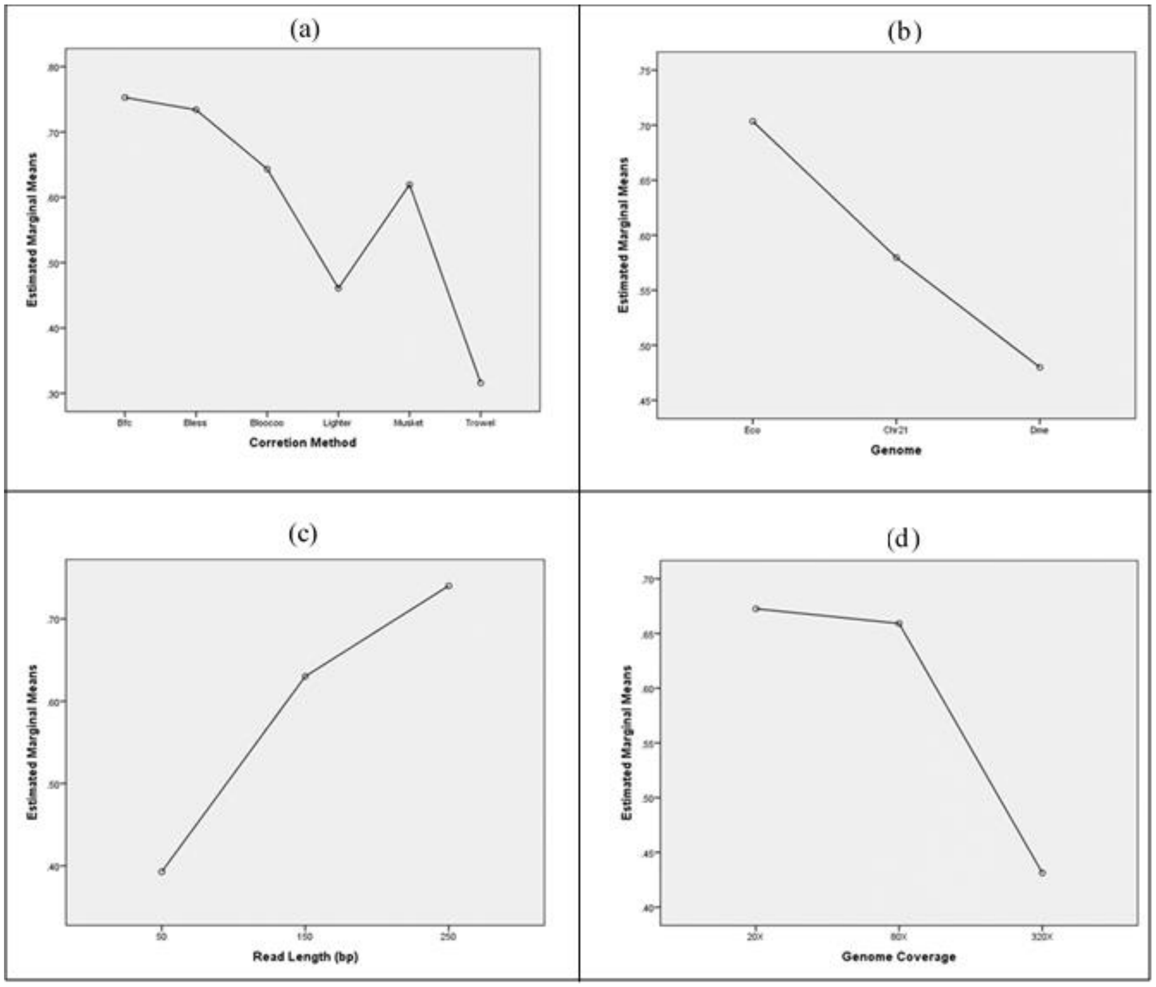


Figure 3.6 The main effects of (a) correction method, (b) genome size, (c) read length, and (d) coverage depth on NGS data correction performance metric F-score with 27 simulated Illumina datasets. Eco = *E. coli*; Chr21 = Human chromosome 21; Dme = *D. melanogaster*

Table 3.8 Statistical significance expressed as F-test probability for the main and interaction effects of four independent variables on performance metrics (precision and F-score) determined using a four-way ANOVA model. P = p-value, df = degree of freedom.

Dependent Variable	Significant Factor	*df*	Type III Sums of Squares	F statistic	*p*
F-score	Genome (G)	2	1.3546	17.63	<.0001
	Coverage (C)	2	1.9849	25.83	<.0001
	Length (L)	2	3.4061	44.32	<.0001
	Method (M)	5	3.8528	20.05	<.0001
	G × M	10	0.9954	2.59	0.0070
	C × M	10	1.0405	2.71	0.0050
	L × M	10	0.9988	2.60	0.0069
Precision	Genome (G)	2	2.3468	30.34	<.0001
	Coverage (C)	2	1.7245	22.30	<.0001
	Length (L)	2	2.0184	26.10	<.0001
	Method (M)	5	0.3471	1.80	**0.1206**
	G × C	4	0.6234	4.03	0.0045
	G × L	4	0.8551	5.53	0.0005
	G × M	10	0.8831	2.28	0.0187
	C × L	4	0.3816	2.47	0.0497
	C × M	10	0.4456	1.15	**0.3320**
	L × M	10	0.6892	1.78	**0.0735**
	G × C × L	8	0.8724	2.82	0.0073

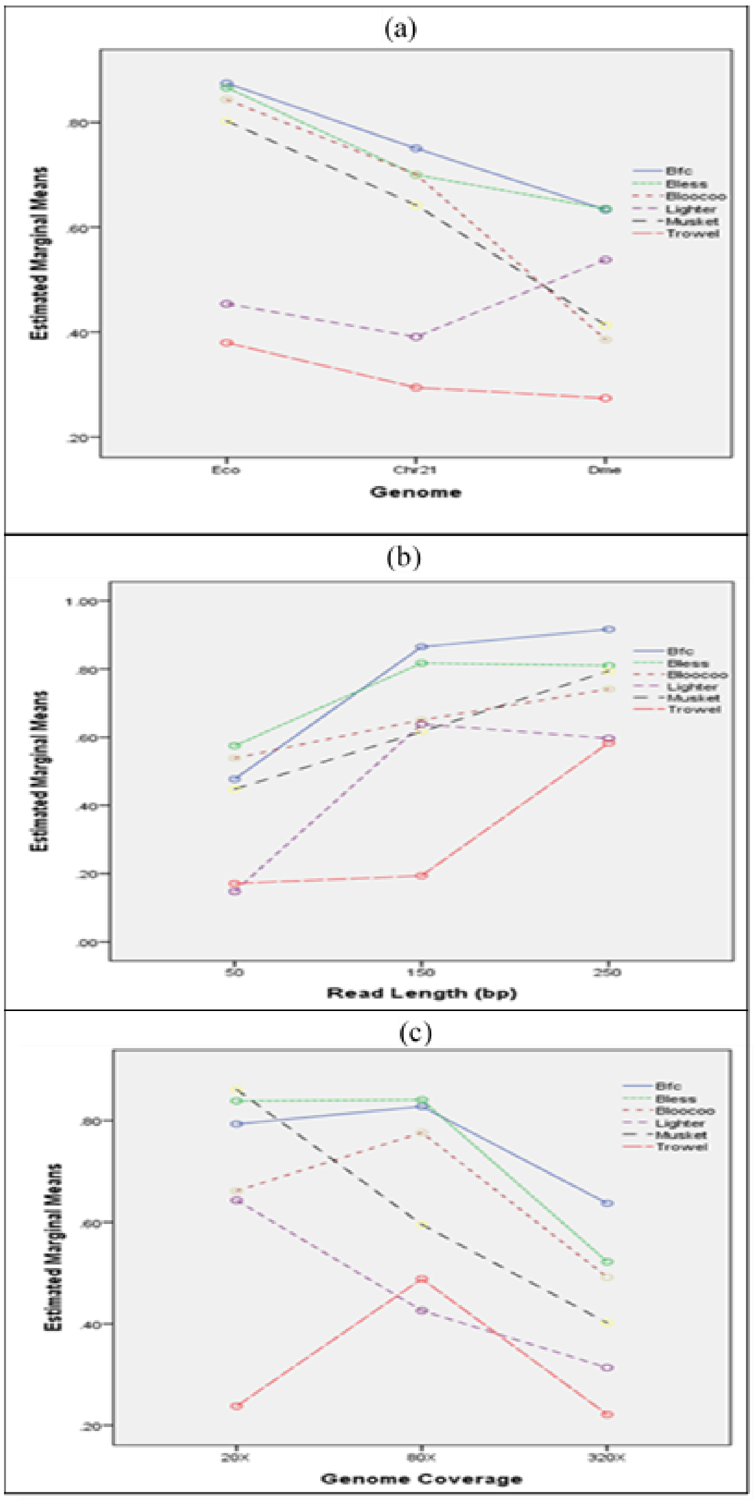


Figure 3.7 Interaction effect of (a) genome size, (b) read length or (c) coverage depth with correction method on F-score

3.6.4.3 Three-way ANOVA Model

Results of three-way factorial analyses are shown in Table 3.9. Genome, coverage, and length all significantly impacted F-score and precision of Musket. BFC, BLESS, and Trowel responded less sensitively to the three factors with only one exception per method, i.e., coverage on precision of BFC, genome on precision of BLESS, and genome on F-score of Trowel. Lighter and Bloocoo were the least sensitive methods. Significant interaction effects were detected only in five instances for F-score, i.e., genome × length for BFC, genome × coverage for BLESS, genome × length and coverage × length for Lighter, and genome × coverage for Musket. Except coverage × length for Lighter, the other four significant instances also occurred to precision. No 3-factor interaction effect (genome × coverage × length) was observed. Read length had significant impact on both precision and F-score of all correction methods (except Bloocoo), whereas coverage depth showed the same effects on F-score but less pronounced effect on precision (insignificant for BFC, Bloocoo and Lighter). Genome size affected F-score and precision of four correction methods.

Table 3.9 Statistical significance expressed as F-test probability for the main and interaction effects of three independent variables (genome, coverage, and length) on each correction method’s performance metrics determined using a three-way ANOVA model. Empty cells indicate statistical insignificance (p > 0.05)

Effect	BFC	BLESS	Bloocoo	Lighter	Musket	Trowel
F-score						
Genome (G)	0.0021	0.0004	0.0093		0.0006	
Coverage (C)	0.0084	<.0001		0.0002	0.0001	0.0017
Length (L)	<.0001	<.0001		<.0001	0.0019	<.0001
G × C		0.0011			0.0328	
G × L	0.0022			0.0032		
C × L				0.0448		
G × C × L						
Precision						
Genome (G)	0.0024		0.0040		0.0009	0.0013
Coverage (C)		0.0012			0.0032	0.0117
Length (L)	<.0001	0.0046		0.0266	0.0493	0.0336
G × C		0.0302			0.0227	
G × L	0.0003			0.0438		
C × L						
G × C × L						

Table 3.10 Reproducibility of KmerGenie-generated optimal k-mer size for two test datasets, DM_80X_L250 and DM_320X_L250

KmerGenie version	DM_80X_L250	DM_320X_L250
1.6476 (initial run)	134	127
1.6982 (repeat-1)	135	130
1.7044 (repeat-2)	143	141

3.6.5 Discussions

Choosing an appropriate NGS reads error corrector could be a daunting job, given the fact that so many choices are available but no systematic comparison has been made so far. As mentioned earlier, many factors may have to be considered. However, from a research point of view, performance in correction accuracy (i.e., how many true erroneous bases are corrected and how many false erroneous bases are mistakenly corrected) is more important because this determines the quality of the corrected data, which is directly linked to downstream analyses. Besides, other factors such as memory consumption and CPU time are resolvable as faster machines with larger RAMs become more affordable or accessible. Hence, we focused on accuracy-related performance metrics.

Two approaches are often employed for error correction performance evaluation: direct and indirect [145]. Direct evaluation assesses the point correction accuracy of single erroneous bases, whereas indirect evaluation assesses correction impact on downstream application (e.g., contig or genome assembly quality, and single nucleotide polymorphism or structural variant call accuracy). We adopted the direct evaluation strategy and point correction-based performance metrics in this study. Since simulated datasets were used and no further genome assembly or other indirect evaluation was performed, we did not develop new performance metrics such as read depth or breadth gain and k-mer depth or breadth gain [145], nor did we use assembly or contig quality metrics [119].

Although limited comparative evaluations have been conducted by the authors of most published correction methods, little information is available on how NGS data characteristics affect method performance. Previously, we used six datasets with varying coverage depth (10X, 20X, 50X, 70X and 120X), genome size (4.6 Mb, 5.4 Mb and 143 Mb), and read length (36 bp, 56 bp and 100 bp) to compare six correctors [140]. Due to the small number of samples (datasets) and a relatively large number of variables, conclusion made from that study has a limited statistical power.

In the present study, we designed a full factorial simulation experiment where correction performance metrics were statistically analyzed for the significance of main factor and interaction effects caused by correction method, genome size, coverage depth, and read length. All factors showed significant influence on performance. Overall, BFC, BLESS, Bloocoo and Musket performed better than Lighter and Trowel (Figure 3.6a); correction of smaller genomes yielded higher F-scores (Figure 3.6b); longer reads led to better correction (Figure 3.6c); but an excessively high coverage depth jeopardized correction performance (Figure 3.6d). It is widely accepted that infrequent errors can be corrected using many other reads covering the same genomic locus only if the sequencing coverage depth is sufficiently high [75], [146], [147]. However, no studies have defined such a sufficiently high level of coverage depth. Our study shows that the 320-fold coverage was so high that it adversely affected the performance of all six correction tools, suggesting that this might be a shortcoming of k-mer spectrum-based correction algorithms. Although method interacted with all other three data characteristics factors (Figure 3.7), it appears that interaction effects were less pronounced than the main effects (Table 3.8). For each individual method, the three dataset characteristics exhibited differential influences as demonstrated by the results from 3-way factorial ANOVA Table 3.8 and Table 3.9). BLESS and Musket appeared to be the most sensitive while Bloocoo the least sensitive to variations in dataset characteristics.

Dataset DM_320X_L50 is clearly an outlier as all six tools performed consistently poor. We repeated one of the six tools (i.e., BFC) on this dataset and reevaluated its performance using ECET. Results (see table 2 http://pinfish.cs.usm.edu/dissertation_tables/metrics.pdf) indicate that the poor error correction performance is reproducible. To explore if the consistent and reproducible poor performance of this outlier dataset was a result of uneven genome sampling, we created another DM_320X_L50 dataset and ran error correction using BFC. Correction performance results (see http://pinfish.cs.usm.edu/dissertation_tables/metrics.pdf) demonstrate that little difference exists between the original and the new dataset in their evaluation metrics. Such an outlier may have been resulted from the joint effects of high genome complexity of the fruit fly, short read length (50-bp, the shortest among the three lengths) and excessively high coverage depth (320-fold). A new iterative strategy of error correction called String Graph Assembler-Iteratively Correcting Errors (SGA-ICE) [147] was published after the current study was completed. SGA-ICE takes advantage of a combination of multiple rounds of k-mer-based correction with increasing k-mer sizes and a final round of MSA/overlap-based correction. We ran it on two datasets (C21_80X_L250 and EC_80X_L250) and confirmed that SGA-ICE outperformed all test tools, including the best performer BFC among them (see Supplementary Table 2 for performance metrics results). However, SGA-ICE did not significantly improve performance metrics over BFC, likely because the performance of BFC was already superb (>0.98 for both precision and F-score). Therefore, there was little room for improvement with these two datasets. Nevertheless, such an iterative strategy can be applied to many error correction tools other than SGA [148], such as Musket [120], to further improve correction performance.

Some caveats are worth mentioning because of several measures we took for convenience and other reasons. KmerGenie was used to generate the best recommended k-mer for each dataset because quite a lot of methods require a user-defined k-mer as input (e.g., Bloocoo) while other methods (e.g., BLESS) automate the k-mer selection or optimization process. It has been reported that k-mer size selection affects the correction performance for some methods while others like Lighter are less sensitive to k-mer size [124]. To save computational time dedicated to k-mer selection and for the fairness of comparison, we pre-selected a KmerGenie-derived k-mer for each dataset (Table 3.4). In principle, the higher the coverage depth, the larger the optimal k-mer size [138]. However, there were two exceptional cases observed in this study: (1) the optimal k-mer size for the DM_80X_L250 dataset is slightly larger (7 bases) than that for the DM_320X_L250 dataset, and (2) the optimal k-mer size for the DM_80X_L50 is slightly smaller (3 bases) than that of the DM_20X_L50 dataset. We repeated the first case by running those two datasets through two later versions of KmerGenie and confirmed its reproducibility (2 to7 bases larger as shown in Table 3.10). A plausible explanation for such a deviation is that the KmerGenie statistical model is prone to some degree of randomness (R. Chikhi, personal communication). Given a dataset of sequencing reads, KmerGenie estimates the optimal k-mer length for genome de novo assembly. It first computes the k-mer abundance histogram for many k sizes, then predicts the number (N) of distinct genomic k-mers in the dataset for each k value, and finally returns the k-mer length that maximizes N [138]. It is possible that the Ns are very similar for a wide range of k-mer sizes, which is reflected to a certain degree in Table 3.10 because the optimal k-mer size varies from 134 to 143 for DM_80X_L250 and from 127 to 141 for DM_320X_L250, with an overlapping range of 134 to 141 bp.

For error correction, it has been shown that with a high enough coverage, selecting k-mers that span about 67% of the sequence read length is desired [147]. This is to allow for error correction to propagate through the middle of the sequence (i.e., correct errors that occur in the middle). For practical purposes, it is advised that it should not be much more than half the read length because longer k-mers improves error correction until the k-mer depth becomes too low. Also, long k-mers allow correction of errors due to low complexity and repetitive genomic elements [146] which often cannot be resolved using short k-mers. For each method, we used the default settings without attempting to optimize other parameters or options. We purposely simulated all three types of errors at an equal rate to test the versatility of a correction method. This might disadvantage some methods that are designed to correct a specific type of errors (e.g., PyroNoise [149])

3.6.6 Review and Future Directions

In summary, we presented a full factorial study that statistically assessed the impact of four factors (i.e., correction method and three NGS data features) on correction performance. Evidence from this study demonstrates that correction performance is highly dependent on these factors. No examined method is free from effects (either main or interaction) of data characteristics, even though some (e.g., Bloocoo) are more resistant than others. Read length and coverage depth had more significant impact on correction performance than genome size. Although only Illumina sequencing datasets were used in the present study, our findings and conclusions can be extrapolated to data generated by other similar NGS platforms. We understand that superior performance on simulated data does not translate 100% to good performance on wet lab-generated experimental data (or real data). The tools selected for the present study have been demonstrated to perform well on both real and simulated data. There are certainly many other factors that may affect tool performance, including the unknown and uncontrollable factors associated with real data. These factors are more challenging to investigate and are beyond the scope of this work. Future studies and new experimental data are warranted to validate the results produced in this simulation study. Additionally, an in-depth comparison of k-mer spectrum-based tools with tools representing other categories of error correction algorithms (e.g., multiple sequence alignment, suffix trie/array, and probabilistic modelling ([75], [146], [150]) may produce broader insights on how algorithmic difference affects correction performance [145].

3.7 Chapter Synopsis

In this chapter, we

- Developed an experiment for a comparative analysis of k-mer spectrum based error correction methods including Lighter, Reptile, Bloocoo, BLESS, BFC, Musket and Trowel.
- Presented an expansion of an earlier comparative analysis from a statistical viewpoint to observe interaction effects of factors, read length, genome size, coverage depth and the error correction algorithms, on error correction performance.

- Identified factors that limits existing error correction algorithms

Our observations indicated that

- Performance of different error correction methods varies depending on the type of NGS data been corrected
- Some error correction methods can perform error correction even with ambiguous bases while others choose to ignore those ambiguous bases and discard reads containing them.
- While some of the methods also use the phred quality score as part of their correction process, others depend solely on the kmer count frequency and error profile generated to correct a given NGS data.

Based on these findings, we conclude that

- Methods such as BFC, Bloocoo, BLESS and Musket performed better than Lighter and Trowel
- BLESS and Musket appeared to be the most sensitive while Bloocoo was the least sensitive to variations in dataset characteristics
- Correction of smaller genomes yielded higher performance overall
- Longer reads led to better correction
- Excessively high coverage depth jeopardized correction performance due to repeats occurring in the data
- A full factorial study can statistically assess the impact of several factors (i.e., correction method and NGS data features) on correction performance.
- No examined method is free from effects (either main or interaction) of data characteristics, even though some (e.g., Bloocoo) are more resistant than others.

- Read length and coverage depth had more significant impact on correction performance than genome size.

CHAPTER IV BIOINFORMATICS ERROR CORRECTION WORKFLOW

4.1 Overview

Complexity of command line based bioinformatics tools presents a monumental challenge to users, especially molecular biologists who do not possess the necessary training to use such tools. This has led to increased availability of web-based frameworks over the years. However, there is no web-based tool for error correction of sequencing reads generated by Next Generation Sequencing (NGS) technologies. With the significance of error correction to downstream analysis, it is imperative that such a web based Graphical User Interface (GUI) be provided to further alleviate the problems facing users when they run command-line error correction tools. The goal of BECOW (Bioinformatics Error Correction Workflow) [http://pinfish.cs.usm.edu/becow] is to alleviate these complexities. BECOW's development was a result of the evaluation performed in CHAPTER III. Having made some important recommendation to users of error correctors, the best performing methods evaluated were incorporated into this pipeline. This chapter describes the main features of BECOW, how it can be used for error correction and results of testing on actual dataset.

4.2 Hardware Environment

BECOW is a web-based error correction workflow with a point-and-click interface that makes it easy to use and provide access to actionable information from the error correction statistics it generates. It is smaller in scope in comparison to some platforms e.g. Galaxy platform but robust enough to handle large sequencing data. Developed using open source software, it runs on a 64-bit Centos Linux server with 16 Intel® CPUs at 2.53GHz, 7Tb of storage space, 244Gb RAM. The pipeline was developed in python and its source code is shown in Appendix 2. The python backend has also been made available on https://github.com/AisaacO/Becow_scripts and distributed under the GPL free software license.

4.3 Background Architecture

Four main error correction algorithms constitute the bedrock of BECOW. These algorithms were chosen for integration because of their performance in correcting a large variation of datasets. The general framework of BECOW is shown in Figure 4.1

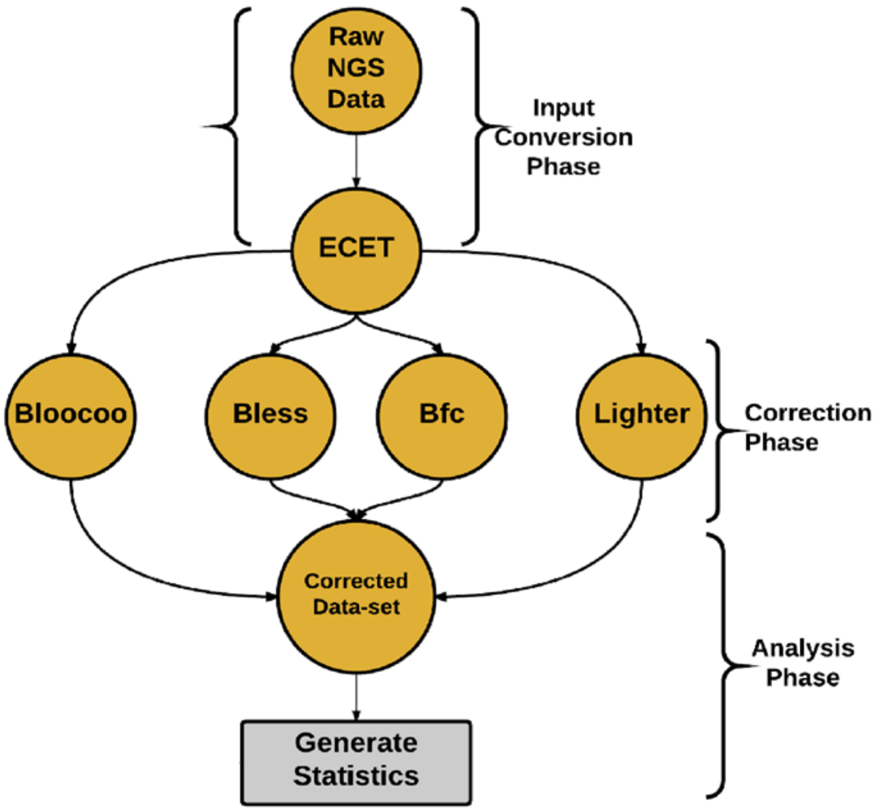


Figure 4.1 shows the general framework of how Becow interacts with the algorithms and submitted data

From Figure 4.1, the Input conversion phase involves converting the files into an independent format TEF. This format allows for an unbiased comparison of the performance result. The correction phase involves the utilization of the error correction methods to correct erroneous bases in the sequence data. Finally, in the analysis phase, comparison is performed to generate statistics about the performance of each of the methods on the input data.

Once user data together with specified parameters are transmitted over the web, the processing follows the steps outlined in the error correction analysis workflow which is as follows:

- The designated data for error correction is aligned to the submitted reference genome. This allow for a precorrection file to be generated in the TEF (Target Error Format) of the ECET (Error Correction and Evaluation Toolkit)
- Next, the available error correction algorithms in the pipeline (BLESS, Bloocoo, BFC and Lighter) begins correction of the submitted NGS data file.
- Once correction is complete, the corrected files are passed through ECET to generate the TEF file of the corrected data.
- The precorrection and corrected alignment file in the TEF format are compared against each other to identify the differences.
- Finally, the differences are generated in the form of performance metrics TP (True Positive), FP (False Positive), FN (False Negative), Recall, Precision and gain.

The behavior of BECOW is controlled by a PHP code which transmits input data to a python code used to integrate all tools that corrects and generates analysis for the corrected data and corresponding error correction method Figure 4.2

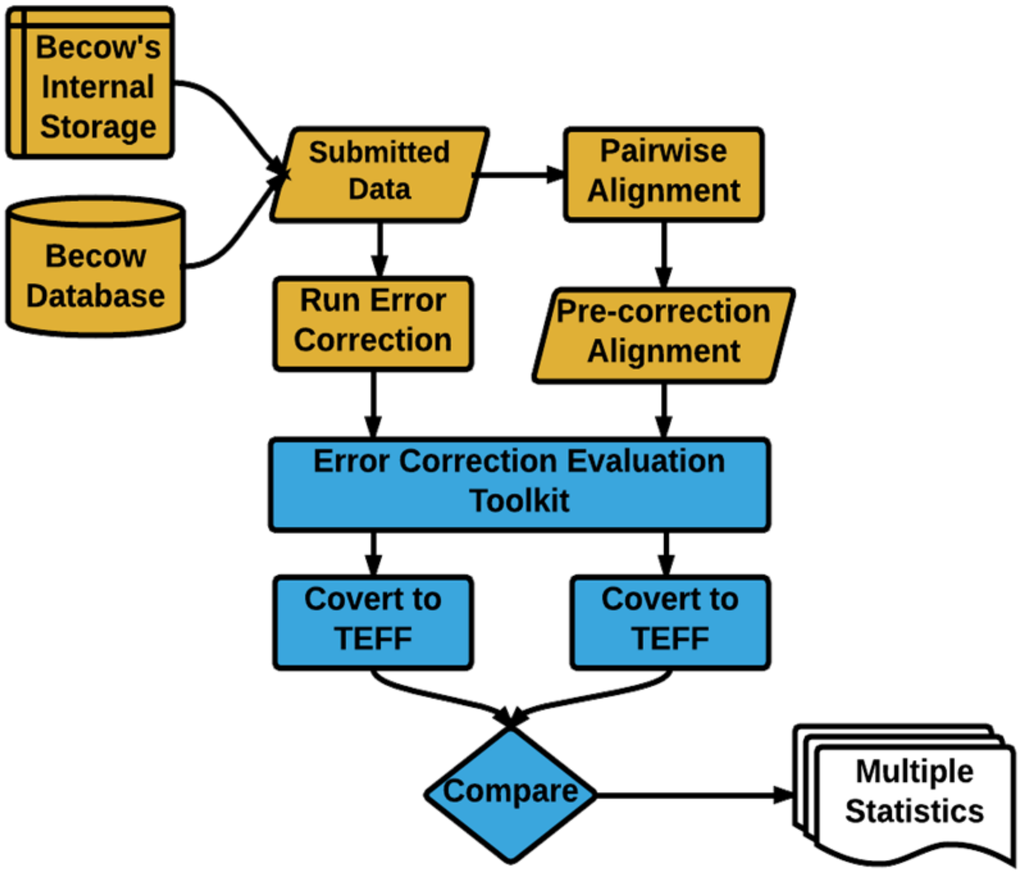


Figure 4.2 displays the python-based backend information flow of Becow's error correction and statistics generation process.

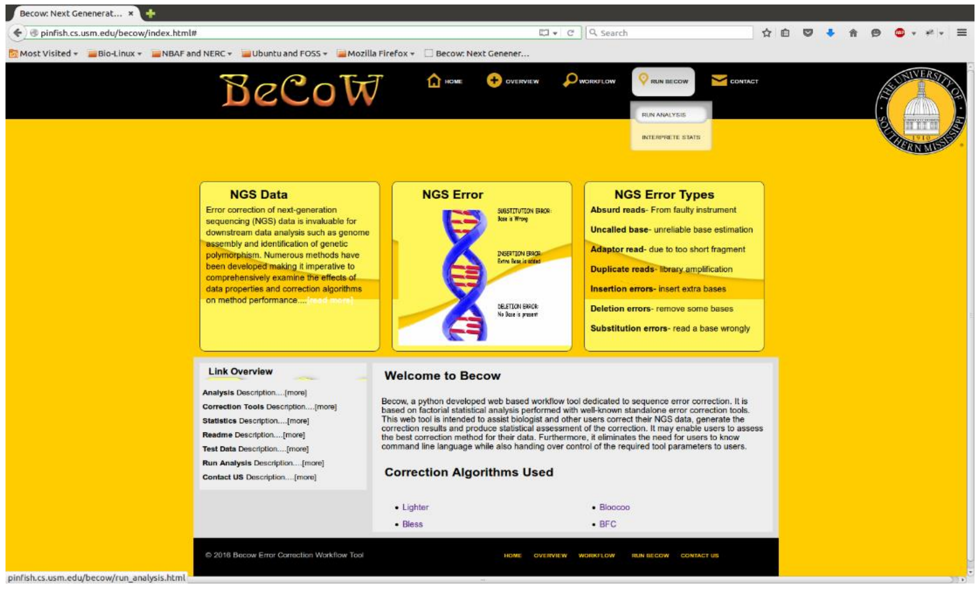
For each submission, the data is stored in its internal storage on The University of Southern Mississippi School of Computing pinfish server. The stored data is indexed in the database for each submission. A user is given a submission identification number.

This number helps in avoiding submission conflict allowing correct identification of the user. Once the data is deemed to be of the right format, the correction and statistics generation process begins following the steps outlined above. The process is visible in the workflow of Figure 4.2. Upon completion, multiple statistics are generated in addition to the corrected data. The statistics will allow a user to determine which corrected data to select for their analysis. This is because performance of error correction methods is dependent on the type of input data. i.e. read length, genome size, genome coverage, etc.

4.4 Web Implementation

The front-end or GUI visible to the user was designed in a simplistic fashion as shown in Figure 4.3. Figure 4.3(a) displays the homepage of BECOW with links to more information about the methods implemented for the error correction, statistical analysis performed, a readme on parameter selection, contacting the author and submission form for running an error correction analysis. There is a seamless transition from one BECOW page to another. Content of the page were styled using CSS. Figure 4.3 (b) displays the form for running an analysis.

(a)


BeCoW
HOME
OVERVIEW
WORKFLOW
RUN BECOW
CONTACT
RUN ANALYSIS
INTERPRETE STATS
SUBSTITUTION ERROR: Base is Wrong
INSERTION ERROR: Extra Base is added
DELETION ERROR: No Base is present

NGS Data

Error correction of next-generation sequencing (NGS) data is invaluable for downstream data analysis such as genome assembly and identification of genetic polymorphism. Numerous methods have been developed making it imperative to comprehensively examine the effects of data properties and correction algorithms on method performance....[read more]

NGS Error

NGS Error Types

Absurd reads- From faulty instrument

Uncalled base- unreliable base estimation

Adaptor read- due to too short fragment

Duplicate reads- library amplification

Insertion errors- insert extra bases

Deletion errors- remove some bases

Substitution errors- read a base wrongly

Link Overview

Analysis Description....[more]
Correction Tools Description....[more]
Statistics Description....[more]
Readme Description....[more]
Test Data Description....[more]
Run Analysis Description....[more]
Contact US Description....[more]

Welcome to Becow

Becow, a python developed web based workflow tool dedicated to sequence error correction. It is based on factorial statistical analysis performed with well-known standalone error correction tools. This web tool is intended to assist biologist and other users correct their NGS data, generate the correction results and produce statistical assessment of the correction. It may enable users to assess the best correction method for their data. Furthermore, it eliminates the need for users to know command line language while also handing over control of the required tool parameters to users.

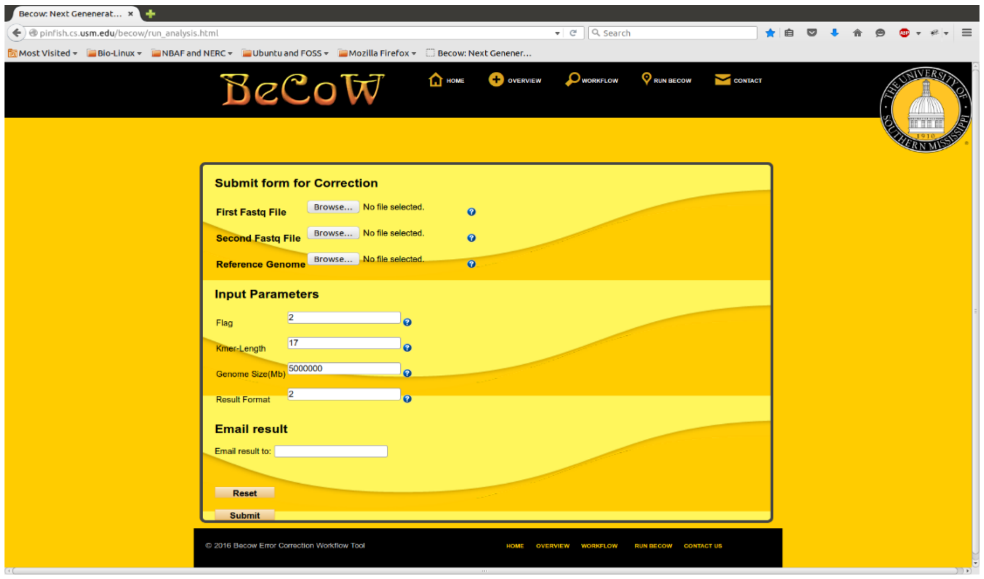
Correction Algorithms Used

- Lighter
- Bless
- Bloocoo
- BFC

© 2016 Becow Error Correction Workflow Tool

pinfish.cs.usm.edu/becow/run_analysis.html

(b)



Submit form for Correction

First Fastq File	Browse...	No file selected.
Second Fastq File	Browse...	No file selected.
Reference Genome	Browse...	No file selected.

Input Parameters

Flag	2
Kmer-Length	17
Genome Size(Mb)	5000000
Result Format	2

Email result

Email result to:

Reset

Submit

© 2016 Becow Error Correction Workflow Tool

Figure 4.3 (a) shows the homepage of Becow displaying how the form can be accessed to submit user NGS data (b) shows the submission form of Becow with various parameters that can be set by the user and submitted for processing using the backend workflow shown in Figure 4.2

4.5 Functionality

The functionality of the pipeline can be grouped into three distinctive parts; Uploading of NGS data files for correction in fastq format, entering parameters for the error correction process and users email address for obtaining result of the analysis.

4.5.1 Data Upload

To perform an error correction, the pipeline requires a pair of uncompressed fastq files and a fasta formatted reference genome. Uploading a pair of files only requires clicking on the upload button and selecting the desired files from its storage location. BECOW uses File Transfer Protocol (FTP) allowing large files to be downloaded in a fast and efficient manner. With a download speed of 347.80Mbps, a 10GB file size takes only approximately 3.926 minutes on the pinfish server.

BECOW employs a series of checks to ensure that the file is in the correct format. The file will not be uploaded if it is not in fastq format. The forward file must end with a 1. fastq likewise 2. fastq for the reverse file. The desired format is checked immediately upon uploading the file. In general, most of the error correction methods incorporated into BECOW are de-novo based. This implies that they do not require a reference genome to perform error correction. The use of a reference genome is to allow ECET to generate important performance metrics for each error correction method. Error correction will still be performed if a user decides not to be interested in looking at the correction statistics. Even though these error correction methods are de-novo based, performance is dependent on the type of NGS data. For most error correction candidate data with an existing reference genome, it is advisable to upload the reference data. This will ensure correction statistics are generated thereby helping the user to make informed decision on which data to use for their downstream analysis.

4.5.2 Input Parameters

Each of the error correction methods implemented in BECOW requires certain input parameters. Default parameters allocated by the methods have mostly been observed to be sufficient for data correction based on the implementation of their error correction algorithm. Currently, there is a requirement for the user to input four parameters namely; Flag, Kmer-Length, Genome size and Result format.

1. The Flag parameter is relevant to ECET. It accepts three integer values:
    - 1 – which implies a desire to keep only reads containing ACTG (actg). This means any bases with characters other than ACTG or actg are discarded during the analysis.
    - 2 – which is the default value keeps all the reads intact without any changes
    - 0 – which implies converting any ambiguous characters in the sequence e.g. N’s to ACTG or actg in a random manner.
2. Kmer Length parameter is relevant for all error correction methods implemented. Some of the methods have a default kmer length while some also have a maximum kmer length that can be used. The default of 17 was determined to be appropriate for most genomes of small organisms. Overall it ranges from 15 to 32 but can be longer or shorter depending on the data

3. Genome size is used by several of the implemented error correction methods e.g. Lighter and BFC. The size is required to be given in Mega bases (Mb). It should correspond to the genome size of the organism whose dataset is been corrected.
4. Result format refers to the user desired result format of the error correction output. It can be in either one of two formats but not both. This parameter is set to 1 for result output in fasta format or 2, for result output in fastq format. Upon correction, if a fasta format is the desired output, quality scores are stripped from correction result and the resulting fasta formatted file is returned to the user.

4.5.3 Email Result

BECOW transmits all information through user email. Once data has been uploaded and the parameters have been specified, the user finally inputs a valid email. Upon submission, an email with the job requirement parameters are sent to the user. The first email indicates that the job has been successfully submitted and is been processed. When the job is complete, a second email is sent to the user with links to the result. The user clicks on the link and downloads the corrected data files and their corresponding statistical analysis. The method is very efficient because correcting the data takes time hence, the results cannot be immediately made available. The results folder is kept for two (2) weeks after which it is automatically deleted from the pinfish server.

4.6 Features of BECOW

Testing and evaluation of BECOW was extensively conducted based on several factors. Factors such as web browsers, efficiency of the operating systems and other computational resources of the server on which the web application is running on (e.g. multiprocessing, server specifications, etc.) contribute, extensively, to a pleasant user experience when using any web application. These factors and their test results are outlined in this section

4.6.1 Supported Web Browsers

BECOW was tested across multiple browsers to ensure that it runs equally well on all major operating systems that supports modern browsers. The tests were conducted using Microsoft® Internet Explorer® version 9, 10, and 11, Microsoft Edge for Windows® 10 and the most recent stable versions of Mozilla® Firefox® and Google Chrome™. With these extensive browser support, it is expected that using BECOW on the tested browsers will deliver the same functionalities and user experience across board.

4.6.2 Operating System

In general, web application promise operating system independence. As a web application, BECOW can be run on any operating system that runs a web browser with internet connection. No OS specific plugins are required. Furthermore, it can also be run on mobile devices e.g. android even though it was not developed to be used from such devices. The mobile experience is limited because it is not adapted to micro browsers. Although, if the data to be processed resides in the mobile device storage, BECOW can still be run and results will still be sent as email link to the user.

4.6.3 Multi-processing

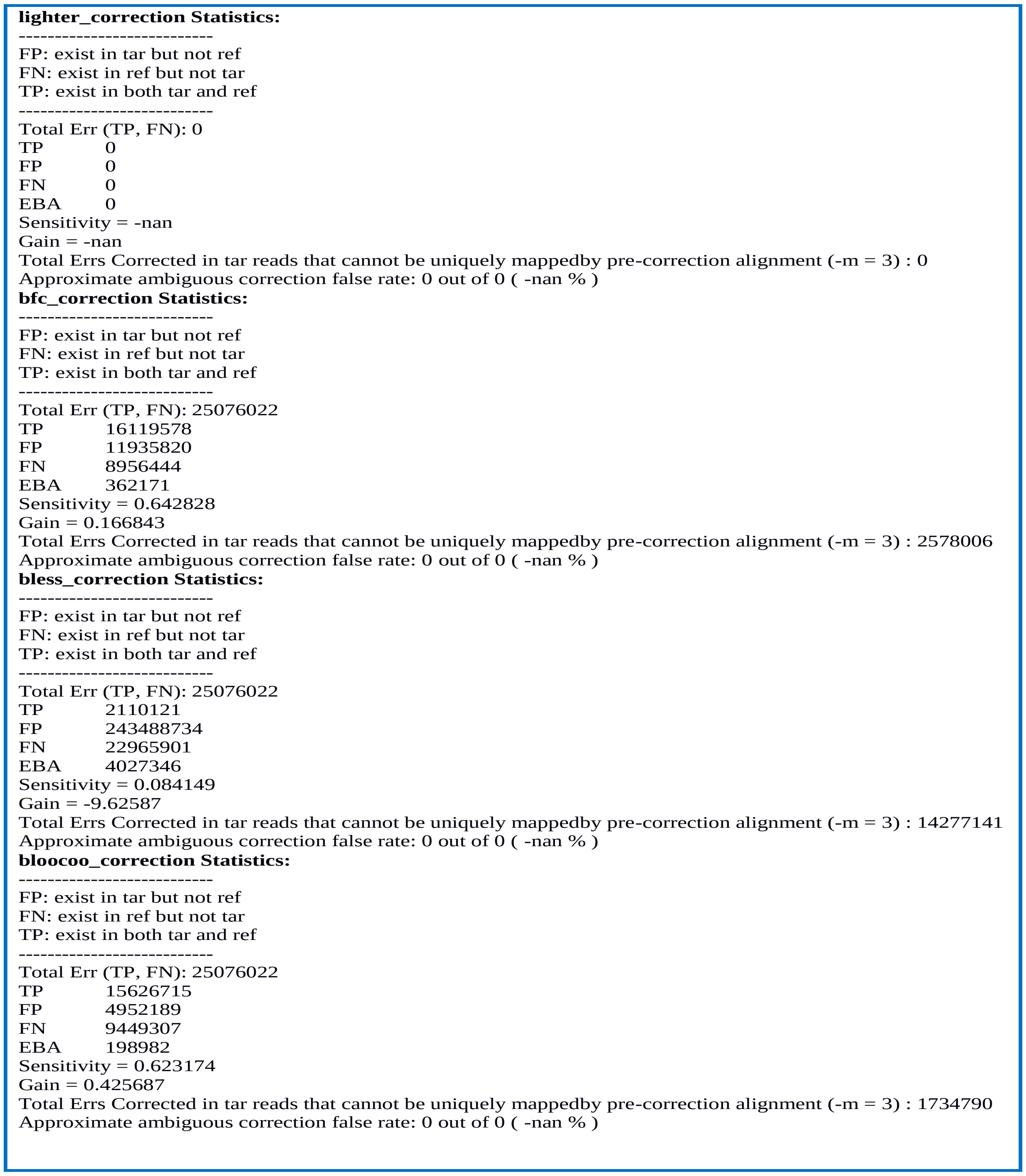
The error correction algorithms implemented provided options to run the analysis with any number of cores, threads, or processors. Pinfish runs on a multi-core processor with 16 threads. BECOW is programmed to run the analysis using all available processors on the pinfish server. This allows for a faster processing of the data resulting in reduced time to completion and result delivery to the user. This implementation saves users a lot of time

4.7 Testing and Evaluation

To evaluate BECOW it was necessary to perform some downstream analysis. The performance of BECOW was tested on *Escherichia coli* illumina data, which is a well-studied organism. After error correction, assembling a genome with a known reference will make it easier to evaluate the performance. The experimental dataset was download from the SRA (Sequence Read Archive) database and extracted in fastq format using the SRA toolkit [151]. At the same time, *E. coli* reference genome (in fasta format) was downloaded from NCBI database like methods employed in CHAPTER III. Both datasets were uploaded to BECOW and its default parameters were used for the error correction. The sample data used can be found on the download sample data link on the webpage of BECOW.

4.7.1 Error Correction Statistics

Upon completion of error correction, BECOW sends an email containing a download link for the error correction results and statistics to the user. The statistics for our sample test evaluating BECOW is shown in Figure 4.4. It contains values for the error correction performance metrics described in CHAPTER III



```
lighter_correction Statistics:
---------------------------
FP: exist in tar but not ref
FN: exist in ref but not tar
TP: exist in both tar and ref
---------------------------
Total Err (TP, FN): 0
TP        0
FP        0
FN        0
EBA       0
Sensitivity = -nan
Gain = -nan
Total Errs Corrected in tar reads that cannot be uniquely mappedby pre-correction alignment (-m = 3) : 0
Approximate ambiguous correction false rate: 0 out of 0 ( -nan % )
bfc_correction Statistics:
---------------------------
FP: exist in tar but not ref
FN: exist in ref but not tar
TP: exist in both tar and ref
---------------------------
Total Err (TP, FN): 25076022
TP        16119578
FP        11935820
FN        8956444
EBA       362171
Sensitivity = 0.642828
Gain = 0.166843
Total Errs Corrected in tar reads that cannot be uniquely mappedby pre-correction alignment (-m = 3) : 2578006
Approximate ambiguous correction false rate: 0 out of 0 ( -nan % )
bless_correction Statistics:
---------------------------
FP: exist in tar but not ref
FN: exist in ref but not tar
TP: exist in both tar and ref
---------------------------
Total Err (TP, FN): 25076022
TP        2110121
FP        243488734
FN        22965901
EBA       4027346
Sensitivity = 0.084149
Gain = -9.62587
Total Errs Corrected in tar reads that cannot be uniquely mappedby pre-correction alignment (-m = 3) : 14277141
Approximate ambiguous correction false rate: 0 out of 0 ( -nan % )
bloocoo_correction Statistics:
---------------------------
FP: exist in tar but not ref
FN: exist in ref but not tar
TP: exist in both tar and ref
---------------------------
Total Err (TP, FN): 25076022
TP        15626715
FP        4952189
FN        9449307
EBA       198982
Sensitivity = 0.623174
Gain = 0.425687
Total Errs Corrected in tar reads that cannot be uniquely mappedby pre-correction alignment (-m = 3) : 1734790
Approximate ambiguous correction false rate: 0 out of 0 ( -nan % )
```

Figure 4.4 Sample error correction performance metrics generated by BECOW

4.7.2 Computational Resource Usage and Speed

For the evaluated *E. coli* dataset, BECOW used a maximum combined RAM of 10.5 GB. A total physical storage of 8GB was used for the combined error correction and analysis process. This includes storage of intermediate results, final results and evaluation statistics. The combined file size of the paired-end fastq file and reference genome fasta file was 2GB. It took a total of 10 minutes from data submission to completion. According to our individual error correction method evaluations from CHAPTER III, this is a total of about 19% decrease in processing time. Making BECOW an effective and efficient web application for error correction.

4.7.3 Evaluation Result Discussion

Genome assembly was performed after error correction. Results from each of BECOW's incorporated error correction method was separately used for the genome assembly. The assembly was performed using MASURCA genome assembler [152]. Default parameters were used and k-mer parameter was kept constant across the board. The result obtained is shown in Figure 4.5. The final draft genome size was used as a measure of assembly performance. This was compared to the reference genome. Though the draft genome size of some of the error corrected data from the BECOW pipeline were lower than expected, overall the accuracy was above average for the *E. coli* dataset. The observation was that the draft genome size generated were comparable to that of the draft reference genome assembly. Factors that may contribute to variation from the reference genome may include:

- Difference in the experimental dataset used in this analysis and that used to generate the reference genome,
- Data QC process used by the publishers of the draft reference genome
- Genome assembly method and tools used to generate the reference genome and
- Finally, the parameters used to generate the reference genome.

One or all the above-mentioned factors, may be the cause of discrepancies in the draft genome assembly result. The evaluation though has shown that BECOW, as a web-based error correction application can generate useable result for downstream analysis.

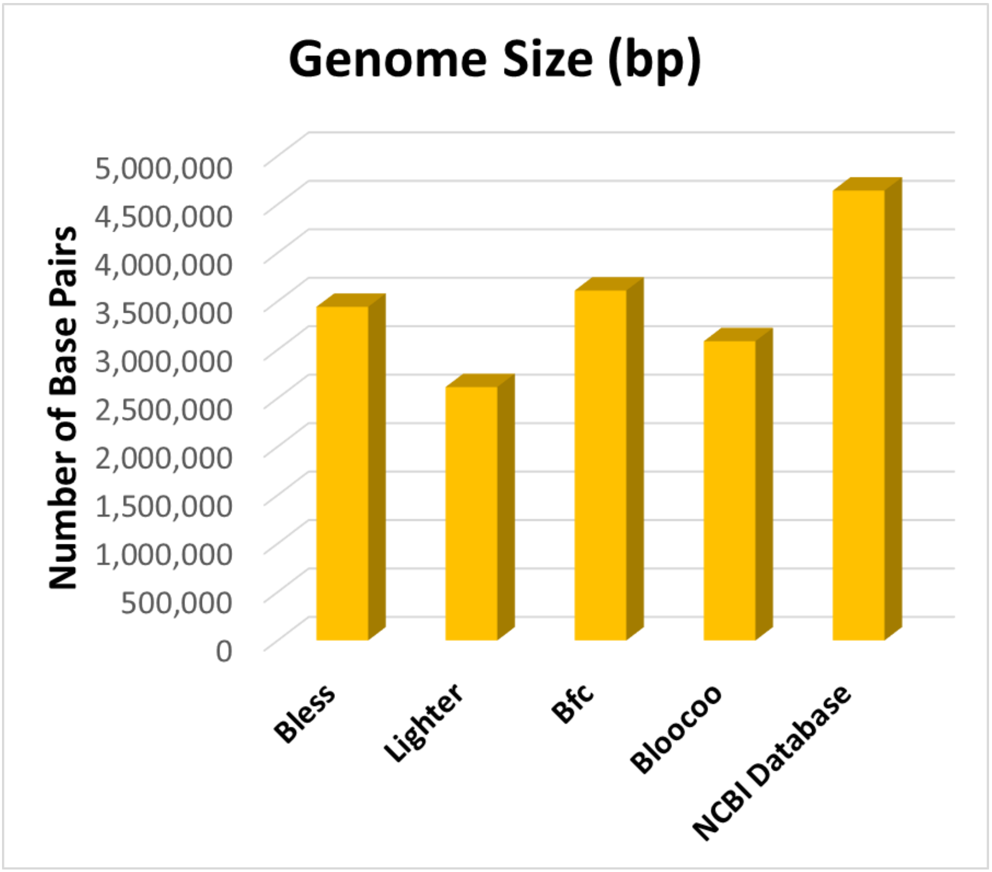


Figure 4.5 Comparison of the draft genome assembly of BECOW's error corrected *E. coli* dataset. Genome size was used as the performance metrics and *E. coli* draft genome assembly as the ground truth for comparison.

4.8 Significant BECOW Contribution

BECOW, as a web-based application for error correction, is highly significant. Though several bioinformatics web-based applications exist, currently there is no application to provide an easy to use GUI for NGS data error correction because they are mostly either stand-alone command-line based or incorporated with other genome assembly tools. The contributions of BECOW are therefore highlighted below:

- Provides a user-friendly GUI method to make error correction easier to perform for any user
- Makes it easy for Molecular Biologists without prior knowledge or training in command-line Linux programming to execute NGS data error correction, which are currently all command-line based
- Provides a means for understanding the errors in the user's NGS data through error correction statistics
- Allows a user, with knowledge of the command-line process, to make informed decisions on the type of error correction algorithm suitable for their NGS data. This allows for further error correction processing if some other user specific parameters are desired for further analysis
- Optimizes the time required by a user to perform error correction of their data using several error correction methods.

- Ensures that a user with limited computational resources available to them can perform error correction of large NGS data without fear of running into storage and computation speed problems.

4.9 Future Directions

The above comparative and statistical analysis of error correction methods and subsequent implementation of BECOW, has led to a deeper understanding of error correction methods. Despite this effort, limitations still exist in the analysis and BECOW's implementation.

Research directions that may be taken in the future, from a comparative analysis effort may include: Extending the recommendations that can be made based on the analysis, Extension of analysis to include other NGS data platforms similar to Illumina and those that generate longer sequence reads e.g. Ion torrent and PACBIO respectively, Extension of analysis to include non-k-mer spectrum based error correction methods e.g. multiple sequence alignment based methods, development of other k-mer spectrum based error correction methods that may be better than existing methods, Expanding the analysis to include a larger variety of experimental dataset for many organisms and developing a more expansive method of performance evaluation.

It is imperative, to include newer error correction methods to further upgrade and update the system as other methods become available, increase allowable data size as input and finally, allow correction of single-end NGS read.

4.10 Chapter Synopsis

In this chapter, we

- Developed a novel web-based NGS error correction workflow as a follow up from our comparative and statistical NGS error correction analysis.
- Showed the significance of BECOW as an NGS error correction pipeline

Our observations indicated that

- Performance of different error correction methods varies depending on the type of NGS data been corrected
- A web-based GUI interface does not exist, currently, for NGS error correction

Based on these findings, we conclude that

- Methods such as BFC, Bloocoo, BLESS and Lighter are excellent candidates for such applications because of their accuracy and frugal resource consumption
- Having a web-based GUI pipeline like BECOW is important to alleviate issues faced by users without the necessary Linux command-line knowledge to run these error correction methods.

We recommend that

- Selection of error correction method for a given type of NGS should be carefully planned based on the characteristics of the data in question
- Finally, optimal parameters, for a chosen error correction method, are important and should also be carefully selected if a qualitative error correction is desired.

CHAPTER V CUCKOO-FILTER ERROR CORRECTION OF NGS DATA

5.1 Background

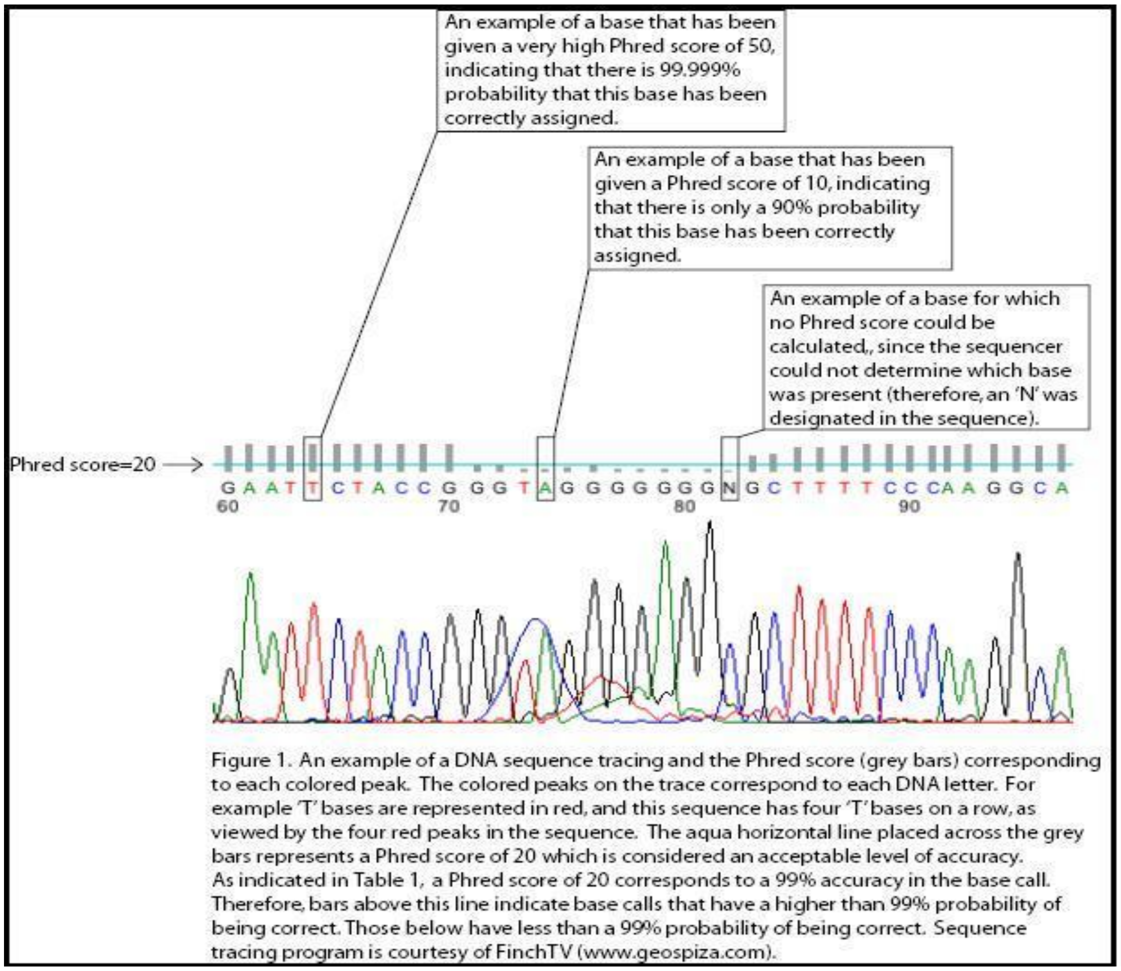
The significance of error correction in NGS data has been discussed in CHAPTER II and a comparative analysis of several existing methods was also elaborated in CHAPTER III. Based on prior discussions and analysis, it was discovered that existing error correction methods possess limitations. Error correction process is laborious and consumes a lot of computational resources. The rate of NGS data generation has continue to rise at a rate higher than Moor’s law [145]. This means that everyone can have access to these data if they desire. Home based users have access to fewer computational resources. Despite the existence of many NGS data error correction methods, the amount of computational resources consumed is not declining even with improved algorithms. Current error correction methods require high computational overhead which is not easily accessible. Therefore, there is a need for error correction algorithms to be space efficient. This minimizes the amount of computational resources used and the time required to perform the error corrections.

Furthermore, the false positive rate of NGS error correction is still high, > 4%, given that most methods evaluated used bloom filters as their data structure. Additionally, engaging in sequence pre-processing steps like read trimming based on length and phred quality score values lead to reduction in sequence length which may produce reads with different lengths. Also, with recent trends towards individual home based DNA analysis, NGS mini-sized sequencers like Minion [153] were developed. These sequencers generate reads of varying lengths. Current k-mer (sequences of consecutive k symbols) based error correction methods target reads of the same length e.g. Illumina based reads. Developing an error correction method that can handle varying read lengths is significant.

Finally, ambiguous bases in NGS data implies there is a base present in that position that could not be called based on inability to assign or calculate the phred quality score Figure 5.1. Several of the analyzed error correction methods do not consider ambiguous bases represented by “N’s” in the data. They either ignore the uncalled bases, discard them, or associate a score value of zero to those bases in their correction process which may be misleading. Consideration must be given to those types of uncalled bases as they may be a source of further arriving at better quality reads. Therefore, there is a significant burden on the research community to develop more precise error correction methods which will facilitate accurate downstream analysis.

CECOND (Cuckoo Filter Error Correction of NGS Data) was developed to target the above-mentioned limitations. Its main goals are efficiency, integrability, generality and usability. The overall idea of this chapter is an attempt to investigate and demonstrate the efficiency and feasibility of using cuckoo filter data structure as an alternative to bloom filters for NGS error correction. We want to indicate that prior to this implementation, no one has tried to use fingerprints of bases for NGS error correction.

This chapter examines CECOND error correction algorithm following this format: First, the steps employed by the CECOND method are outlined. Secondly, the algorithm implementation detail is presented. Thirdly, testing and evaluation of the method in comparison to the most current kmer-spectrum based error correction methods like BFC, BLESS, Bloocco and Lighter, are discussed. Next significant contributions made by CECOND are deliberated. Lastly, the conclusion of the chapter and directions for future research are discussed.



An example of a base that has been given a very high Phred score of 50, indicating that there is 99.999% probability that this base has been correctly assigned.

An example of a base that has been given a Phred score of 10, indicating that there is only a 90% probability that this base has been correctly assigned.

An example of a base for which no Phred score could be calculated,, since the sequencer could not determine which base was present (therefore, an 'N' was designated in the sequence).

Figure 1. An example of a DNA sequence tracing and the Phred score (grey bars) corresponding to each colored peak. The colored peaks on the trace correspond to each DNA letter. For example 'T' bases are represented in red, and this sequence has four 'T' bases on a row, as viewed by the four red peaks in the sequence. The aqua horizontal line placed across the grey bars represents a Phred score of 20 which is considered an acceptable level of accuracy. As indicated in Table 1, a Phred score of 20 corresponds to a 99% accuracy in the base call. Therefore, bars above this line indicate base calls that have a higher than 99% probability of being correct. Those below have less than a 99% probability of being correct. Sequence tracing program is courtesy of FinchTV (www.geospiza.com).

Figure 5.1 Example of DNA sequence tracing and Phred score (grey bars) corresponding to each colored peak as adapted from [154]

5.2 Error Correction Model

Data structure plays an important role in the false positive rate and the amount of computational resources used. Probabilistic data structures, implemented by most existing error correction methods, have a false positive rate of >4% i.e. Bloom filter. Even though they are space efficient, they are not optimal [155]. A more recent probabilistic data structure known as cuckoo filter [156] was discussed in CHAPTER II and has been shown to test set membership with a target false positive rate < 3% . It also possesses a better space efficiency in comparison to the bloom filter [115] used by most error correction methods. Implementing the cuckoo filter data structure may result in lowering the number of false positives generated during error correction. It may also improve space efficiency which will result in reduced consumption of computational resources and possibly, increased speed.

CECOND error correction is also a k-mer spectrum based error correction method developed in C++. Given the large size of NGS data which contains candidates for error correction, it is only reasonable to use a compiled language like C++ for its development because of requirement for faster processing times. The general algorithm workflow of CECOND is illustrated in Figure 5.2. A feature of CECOND include its ability to accept both FASTQ and FASTA formatted sequences for error correction. Once sequence files and options are supplied, CECOND begins by checking the options. The options for the file formats are checked first. If option r = 1 is provided, it indicates that the files are in fasta format and CECOND proceeds directly to correction process starting with stage I. If option is r =2 or option r not stated, it checks the phred quality score values Q, of the sequence to ensure that the average quality is greater than 20 i.e. > 0.1% probability of incorrect base call or 99% base call accuracy. This implies some level of contextual information is required. Once the quality scores are checked but found to be < 20 on the phred score scale, the user is prompted to further pre-process their data to bring the average quality score up to 20 at the least and the process is terminated. If no option is provided but the associated phred quality score is checked and found to be ≤ 20, CECOND proceeds to stage I of the error correction process.

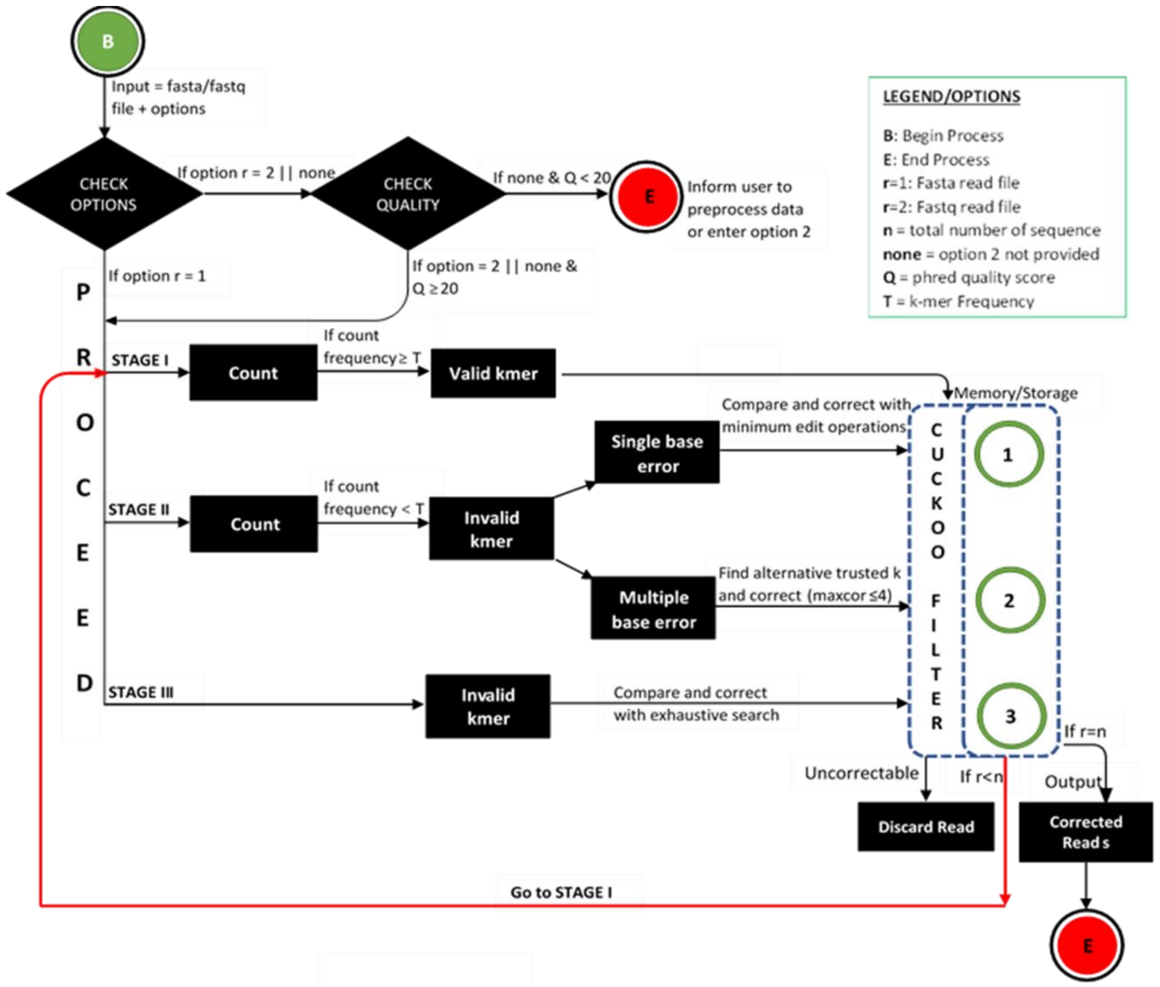


Figure 5.2 The flow chart of CECOND algorithm showing error correction steps. Results from each correction stage is stored and collected at the end for the final error correction result

5.3 Methodology

Prior to beginning of stage I and after checking that all parameters have been properly set, CECOND starts processing by randomly converting all ambiguous “N” nucleotides to one of A, C, T, or G (process not shown in flowchart Figure 5.2). We assume that user supplied data have been properly pre-processed to remove low quality bases also associated with ambiguous bases. The distribution of the number of Ns in the reads, at this point, will be minimal, if any, and will be of high quality. Dealing with ambiguous nucleotides can be complex and may cause significant delay in computation. Discarding reads containing Ns may lead to loss of information. Random assignment of nucleotides to ambiguous bases will help reduce information loss and may aid error correction. A systematic guide on how CECOND was implemented is presented in APPENDIX A

5.3.1 The k-mer Counting Problem

The determination of k-mer (length k substring of a sequence) abundance in genome sequencing finds application in many areas of genome projects. Areas such as de novo assembly [157], detection of repeats in a genome [109], genomic sequence duplication [158], multiple sequence alignment [159] and direct provision of biological insights [160]. It is also used in all NGS error correction methods. The k-mer counting problem is stated as:

Given a group of read sequences R and a substring of R of length K, find the frequency occurrence of K in R. One way to count this k-mer is illustrated in Figure 5.3

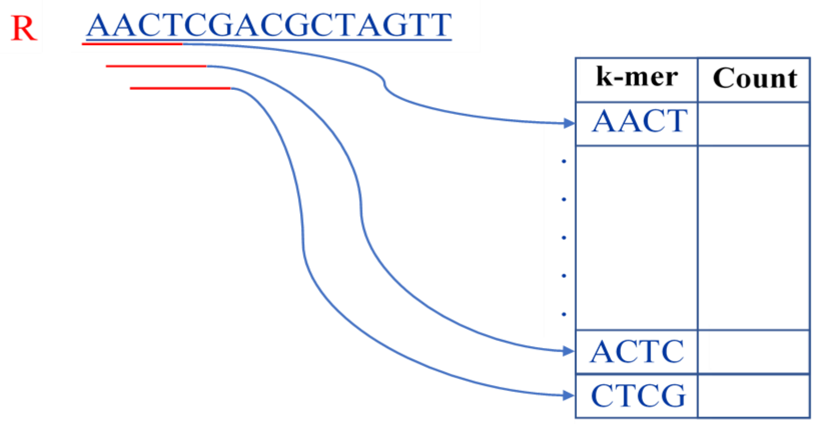


Figure 5.3 Example of a simple approach to k-mer counting (this case implies k=4). Just hashing and collision resolution will take a long time using this method

But with the several giga bases of biological sequence data available for analysis, issues of scalability are encountered since huge amount of computational resource usage is required. The bottleneck in the kmer counting problem lies in how to efficiently count the k-mers to maximize computational resource usage and speed. Since errors in sequencing data generated from sequencing platforms lead to unique k-mers, larger genomes will contain a higher number of those k-mers. This implies erroneous k-mers maybe greater than non-error containing ones. Considering the huge amount of data generated by NGS technologies, several methods have been implemented to tackle the k-mer counting problem [128], [129], [160]–[162] [163].

In general, they all employ some form of data structures like bloom filters, hashing strategies (single or multiple tables) and arrays. While some directly count the k-mers, others use minimizers for sorting the k-mers into super k-mers (substrings of length $\geq$ k which contains k-mers that share the same minimum p-substring) where p $\leq$ k [164].

5.3.2 Counting K-mer Frequency

First stage of CECOND involves counting the k-mer occurrence based on user specified k value of k-mer. A k-mer counting algorithm approach similar to KMC [129] is used. The k-mer counting problem discussed in 5.3.1 Each k-mer present in the read is consecutively extracted starting from the leftmost base until the end of the read. They overlap by k-1. Though devising a means to extract these k-mers is simple, the bottleneck lies in the size of the data. Since the data is not strand specific (unknown orientation) and both a kmer and its reverse compliment are equally likely to be seen, an efficient method is required to conserve memory used during the counting. Multiple copies are eliminated using canonical k-mers. i.e. both k-mer and its reverse complement are considered and the numerical lesser of the two is selected. For a given sequence read R containing bases b eq. (1), the reverse complement Rc is given by eq. (2):

$$R = \{b_1 b_2 b_3 \ldots . b_n\} \in \{A, C, T, G\}^n \quad (1)$$

$$Rc = c(b_n) \ldots c(b_3)c(b_2)c(b_1) \mid c(A){:=}T,\ c(C){:=}G,\ c(T){:=}A\ \&\ c(G){:=}C \quad (2)$$

Hence, the k-mers are first extracted before conversion into its canonical representation. During this phase, k-mers are extracted based purely on the pre-determined k value. To count the k-mers, we used cuckoo filter (see CHAPTER II) with 2 hash tables.

5.3.2.1 Implementation of Cuckoo filter

Cuckoo filter stores the fingerprints of the k-mers observed during the count while the hash table stores all the k-mers with a frequency greater than a pre-determined threshold. The number of possible k-mers increase as the k-mer length is increased. Overall, based on the assumption that there are only 4 types of bases in the sequence i.e. A, C, T, and G, we can estimate the total number of k-mers:

$$T_k = B^k \tag{3}$$

where T_k = total number of k-mers, B = total possible bases and k = k-mer length.

To determine the cuckoo filter hash table size (number of buckets), we first estimate the number of k-mers of a given a read by counting the possible k-mers in a user supplied sequence file (always 4) and k-mer value using eq. (3). Table 5.1 shows total number of bases calculated for different k-mers (up to 10). Knowing the total number of k-mers helps prevent failure due to table filling up or requiring rebuilding, which can be additional (re-insertion) computational cost. Also, since current cuckoo filter allows a maximum of 500 relocation attempts, for insertion into a vacant bucket, having a predetermined table is crucial. Relocation only stops when a vacant bucket is found or it reaches the maximum value. If this happens and no extra vacant position can be found, the filter may fail and the process terminated.

Table 5.1 Total possible k-mers in a sequence with only A, C, T, G bases

Bases	**k-mer Length**	**Total Possible k-mers**
4	1	4
4	2	16
4	3	64
4	4	256
4	5	1,024
4	6	4,096
4	7	16,384
4	8	65,384
4	9	262,144
4	10	1,048,576

Based on estimates from [156], we desired optimal performance therefore we chose a target false positive rate $\mathcal{E}$ that will lie between the range $0.00001 < \mathcal{E} \leq 0.02$ in our corrected data. The default (2,4) cuckoo filter is suitable for this purpose and implies an optimal bucket size was chosen. Meaning, each k-mer $\boldsymbol{k}$ will have two buckets which can contain up to 4 counts as fingerprints in the cuckoo table. It is important that an optimal bucket size be chosen because it is directly proportional to the probability of false fingerprints hit eq. (4) This helps to reduce the space occupancy (amount of memory used). Since our cuckoo filter does not require support for deletion in terms of error correction, the space saved in this manner is sufficient. There was no need to sort the fingerprints using semi sorting cuckoo filter which will further reduce space usage but " requires extra coding/decoding tables and indirections on each lookup" [156].

5.3.2.2 Selecting k-mer Threshold

Before error correction, most k-mer based methods select a specific threshold over which a given k-mer is assumed to be valid. Some use k-mer frequency or coverage distribution histograms to determine the threshold [72], [119], [120], some use quality thresholds [121], some define a range of thresholds (3-6) [122] while others [165] use repetition depth to calculate the threshold. CECOND uses the k-mer count coverage depth to calculate the threshold. Instead of using a single threshold for the entire reads like the others, we calculate the threshold for each read in a way similar to [165]. Figure 5.4 illustrates an example of a k-mer count abundance (distribution) plot.

In this method, for each read, a k-mer count depth histogram is determined, calculate the harmonic mean of the counts and generate a strand specific threshold T using the adjusted mean. The intention here is to reduce the effect of small regions of the sequence which are highly repetitive and using the harmonic mean can negate such effects [165] resulting in the possibility of correcting errors in high regions and minimizing wrongful error correction of reads with low coverage depth. The calculation is performed using eq. (6)

$$\text{Harmonic mean of k-mers } K = n \cdot \left(\sum_{i=1}^{n} \frac{1}{k_i}\right)^{-1} \tag{6}$$

where n is the number of k-mers observed and k_i is the k-mer at the ith index.

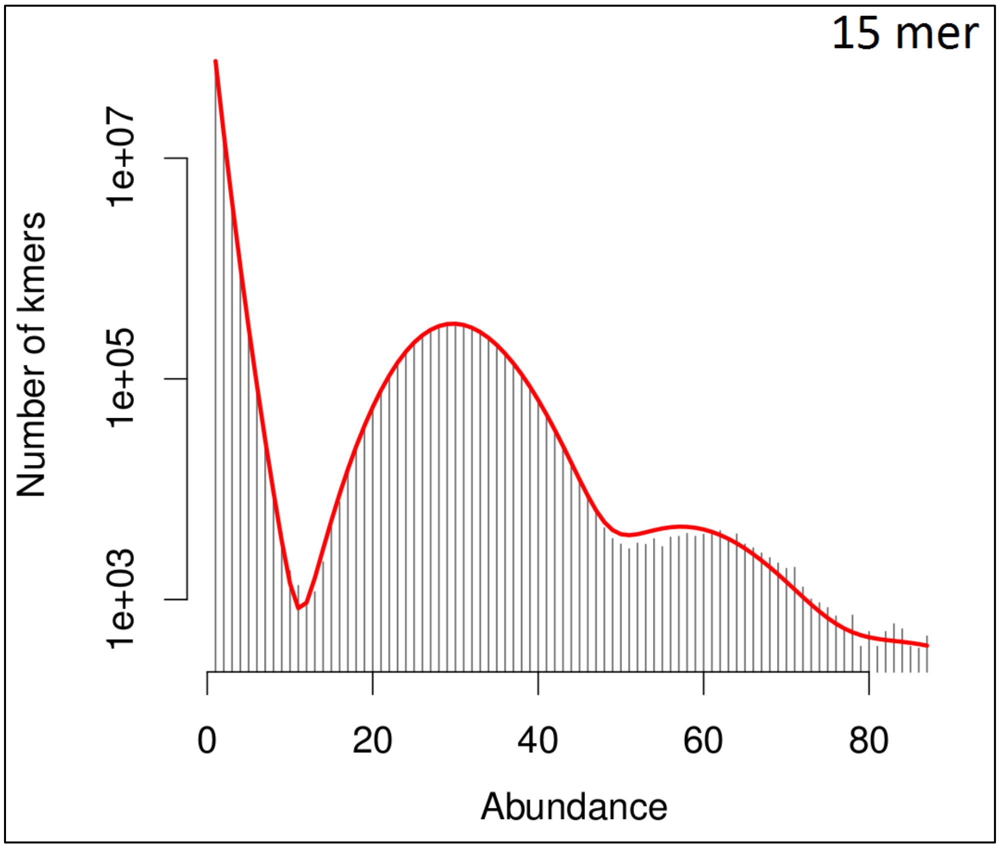


Figure 5.4 Example of a 15-mer count coverage distribution for *Escherichia coli* using k=15 showing the peak at 36. A strong divergence at low kmer frequency is indicative of errors while the peak is the coverage with the highest number of different 15-mers i.e. the average coverage depth is around 36 though the normal like curve observed indicates there are regions with less or more coverage.

To reduce computational resource usage, only most frequently occurring k-mers are stored. CECOND classifies a k-mer as correct based on the auto determined threshold from a generated k-mer coverage histogram as explained above. If a k-mer has a frequency greater than the threshold, the frequency of the count is stored as fingerprints in cuckoo filter rather than key-value pairs like in bloom filter. Each k-mer fingerprint stored is one entry in the cuckoo filter table which can store multiple entries in one bucket. The entries are added dynamically and only one copy of an identified k-mer is stored in the hash table.

5.3.3 Error Correction

Stage II of CECOND involves error correction. First, it corrects single base errors where the base is found to be wrong and secondly, correction of multiple base errors where there exist up to a maximum of four (4) base errors and finally, it checks for more errors and try to correct them using exhaustive search. If a correction cannot be made at that point, the read is discarded.

Single base errors occur most often as sequence specific errors [166] and can be either substitution or mismatch errors (where a base is replaced by another base in the sequence or specifically, single nucleotide is misinterpreted). These errors can be identified using coverage statistics of the most occurring k-mers in a sequence. i.e. they are the bases that are not supported by the k-mer evidence. To correct these type of errors CECOND uses a non-greedy approach. These errors can be identified using the k-mer count profile as illustrated in Figure 5.5 and Figure 5.6.

Counting the k-mer abundance and generating a profile indicates k-mers that occur less frequently and maybe candidates for error correction. As shown, for an 8-mer count, it is expected that the average number of times, any given 8-mer counts occur should be about 8 or more. K-mer counts that are below average create a trough in the k-mer count profile indicative of base errors. A count profile of k-mers is used to detect where these errors lie. CECOND, compares the counts at these troughs to the determined threshold (see 5.3.2.2) and proceed to correct the k-mer if its value is seen to be less than threshold.

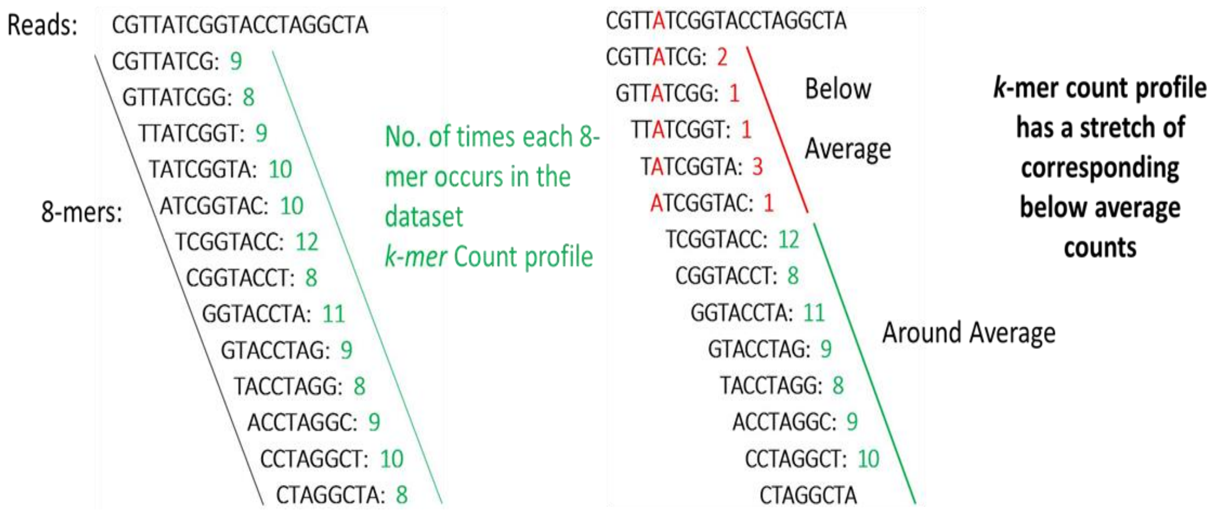


Figure 5.5 identifying k-mer counts for building a k-mer count profile

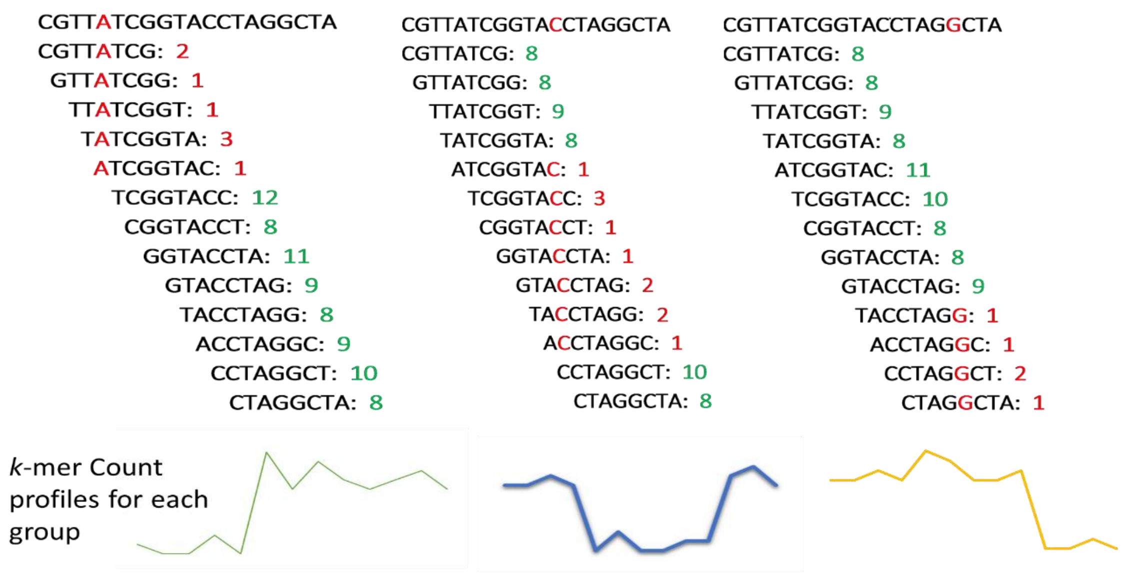


Figure 5.6 Example of k-mer count profiles generated for three different reads. Errors located in the middle of a read generally affects the k-mer count more than errors located at the 3' end of a read.

CECOND proceeds to explore possible corrections by looking at both the coverage threshold and the neighborhood of an identified error containing k-mer for contextual information as shown in algorithm 2. A k-mer's neighbor is its adjacent k-mers. Example illustrated in Figure 5.7 with 3-mers.

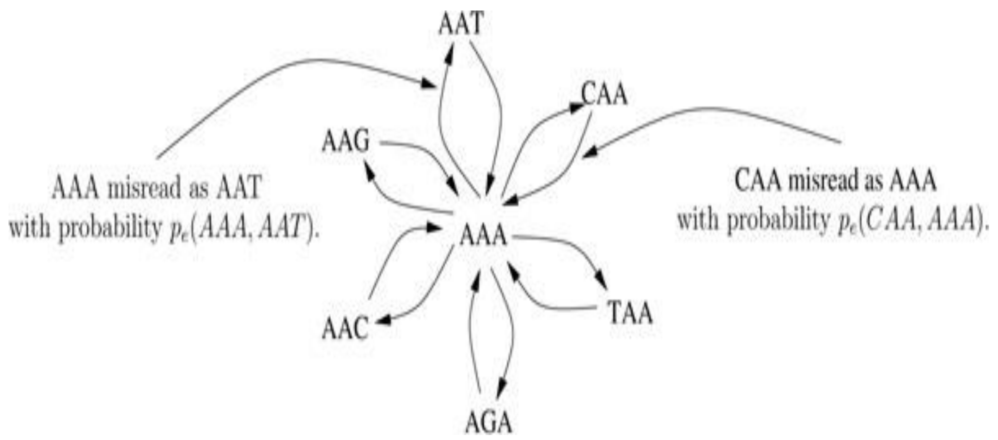
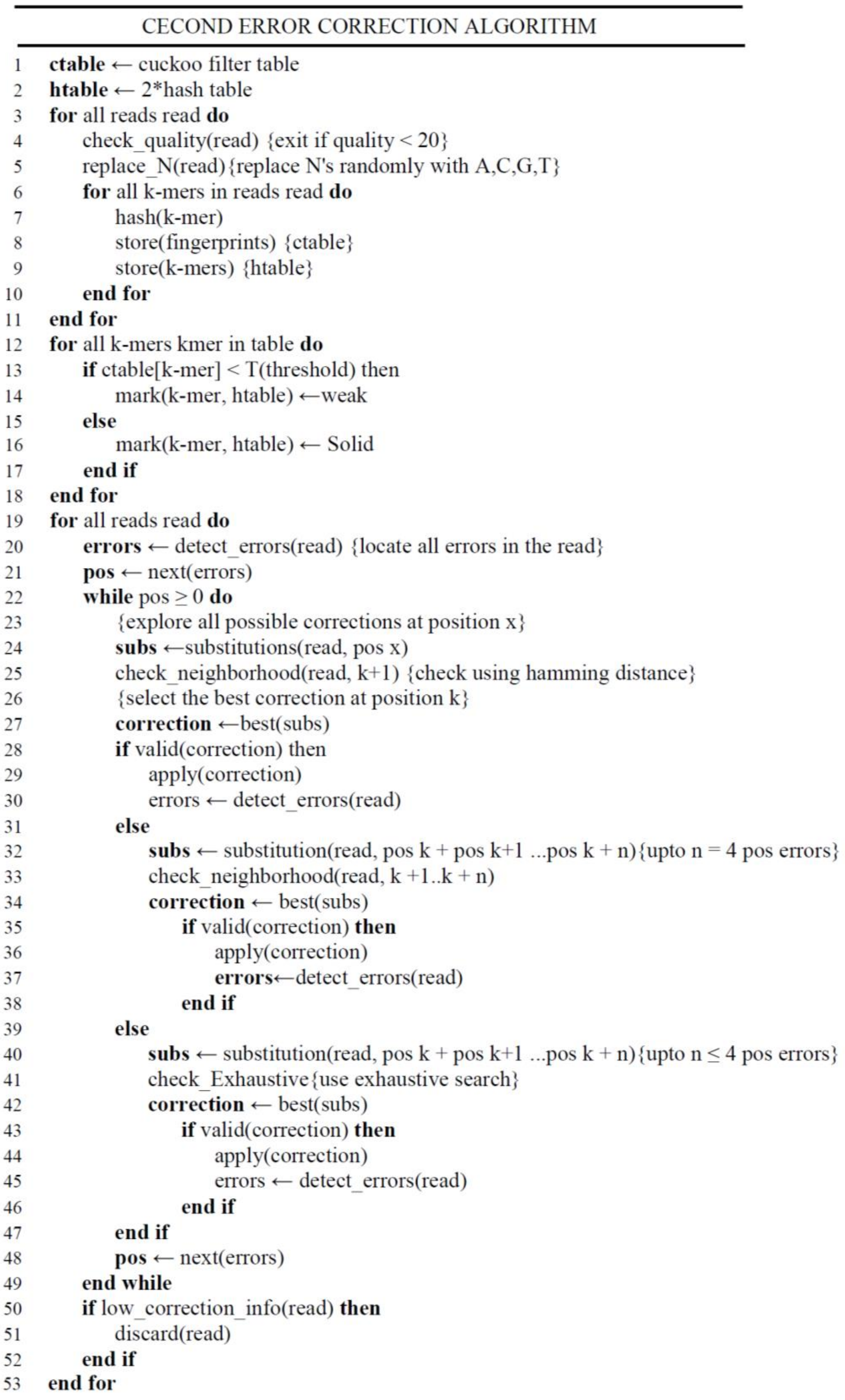

AAT
CAA
AAG
AAA
AAC
TAA
AGA
AAA misread as AAT with probability $p_e(AAA, AAT)$.
CAA misread as AAA with probability $p_e(CAA, AAA)$.

Figure 5.7 A k-mer neighborhood. The neighborhood of trimer AAA is the collection of trimers in R3 that have a non-vanishing chance of being misread as AAA, in this case trimers with at most one substitution. Text and illustration adapted from [167]

It checks the neighboring k-mer of the k-mer in question to determine the number of nucleotide (base) difference between them. It uses a minimum hamming distance of 1, based on hamming codes [168], to check that the difference between the k-mer and its neighbor is 1. If the neighbor is a valid k-mer (frequency count ≥ T, threshold) and the difference is 1, the erroneous k-mer gets modified or replaced with the neighbor k-mer. It does this recursively until the end of file and outputs an intermediate result. This result is used in the next step where multiple errors in a sequence read are corrected. The same method is applied but a maximum correction of 4 bases is allowed. The hash tables are revisited to check for other invalid (erroneous) k-mers. If more k-mers are found to be invalid, we apply exhaustive search to correct the kmer if possible. If not, the read is deemed uncorrectable and discarded. The algorithm of CECOND is shown in Figure 5.8 Certain criteria that allows successful error correction includes:

- Having a high enough k-mer coverage depth
- Setting k to a high enough value to allow differentiation of unique k-mers from frequently occurring k-mers
- The k-mer neighborhood, which is dependent on error rate and k, should be broad enough to allow finding of frequent k-mers
- Use a data structure that will not complicate the error correction process further based on the counts.



CECOND ERROR CORRECTION ALGORITHM

```
ctable ← cuckoo filter table
htable ← 2*hash table
for all reads read do
    check_quality(read) {exit if quality < 20}
    replace_N(read){replace N's randomly with A,C,G,T}
    for all k-mers in reads read do
        hash(k-mer)
        store(fingerprints) {ctable}
        store(k-mers) {htable}
    end for
end for
for all k-mers kmer in table do
    if ctable[k-mer] < T(threshold) then
        mark(k-mer, htable) ←weak
    else
        mark(k-mer, htable) ← Solid
    end if
end for
for all reads read do
    errors ← detect_errors(read) {locate all errors in the read}
    pos ← next(errors)
    while pos ≥ 0 do
        {explore all possible corrections at position x}
        subs ←substitutions(read, pos x)
        check_neighborhood(read, k+1) {check using hamming distance}
        {select the best correction at position k}
        correction ←best(subs)
        if valid(correction) then
            apply(correction)
            errors ← detect_errors(read)
        else
            subs ← substitution(read, pos k + pos k+1 ...pos k + n){upto n = 4 pos errors}
            check_neighborhood(read, k +1..k + n)
            correction ← best(subs)
                if valid(correction) then
                    apply(correction)
                    errors←detect_errors(read)
                end if
        else
            subs ← substitution(read, pos k + pos k+1 ...pos k + n){upto n ≤ 4 pos errors}
            check_Exhaustive{use exhaustive search}
            correction ← best(subs)
                if valid(correction) then
                    apply(correction)
                    errors ← detect_errors(read)
                end if
        end if
        pos ← next(errors)
    end while
    if low_correction_info(read) then
        discard(read)
    end if
end for
```

Figure 5.8 Algorithm for CECOND

5.4 Testing and Evaluation

Testing and evaluating the method is a crucial aspect in determining its effectiveness. CECOND aims to target space efficiency, handle repetitive regions better and improved accuracy. One more feature is its application of cuckoo filter to both kmer counting and error correction. It is known in the research community that k-mer counting constitutes considerable memory overhead given the large size of data to be processed. This means that with the exception of a few k-mer spectrum based methods, several error correction methods make use of existing k-mer counting methods like [128], [129], [160] for their k-mer counting needs required for error correction. Here, we evaluate CECOND in comparison to other existing error correction methods that have been discussed in CHAPTER III

5.4.1 Materials and Method

For this evaluation purpose, we used both synthetic and experimental datasets. Synthetic dataset used are same as discussed in CHAPTER III namely; *Escherichia coli*, Human chromosome 21 and *Drosophila melanogaster* reference genomes. The data was generated using ART at various lengths and sequence coverage depth (see Table 3.4). Testing was conducted using the same parameters, computational environment, and settings as our previous evaluation of the existing error correction methods [169]. For evaluation of CECOND, only a subset of the simulated data was used. (see Table 5.4)

Following evaluation on synthetic data, to further evaluate the accuracy of CECOND, six publicly available (popular for use in evaluating novel error correction methods) and well characterized experimental data were selected, downloaded from the SRA database, and used as benchmark for evaluation. The sequence data, which are all

Illumina based, includes *Staphylococcus aureus* (referred to as S1), *Escherichia coli* (referred to as S2), *Saccharomyces cerevisiae* (referred to as S3), *Caenorhabditis elegans* (referred to as S4), and human chromosome 14 (referred to as S5) see Table 5.2 for the characteristics of the sequence datasets. S1 and S5 were the same sequence data used in the GAGE (Genome Assembly Gold-standard Evaluations) competition [170] and were also used in the evaluation of BLESS [119]. S4 is the same dataset used during BFC [123] and BLESS 2 [142] evaluations.

Based on our inability to directly measure performance on experimental dataset, pre-correction and post-correction alignments were performed. Statistics generated from the alignment, were used to measure performance. Mapping the corrected reads to a reference genome and counting the number of mismatches is an effective method used regularly to evaluate error correction methods. We used BWA [171] for this purpose. Reads containing ambiguous bases were removed for a fair comparison and to ensure that evaluation by ECET [75] is accurate. ECET avoids N's in a read so it is imperative to remove them.

Table 5.2 Experimental datasets downloaded from SRA database and their characteristics

Label	Genome	Number of Reads (Mb)		Accession Number		Genome size	Read length	Cov. (X)
		Pre-processing	Post-processing	Reference	Read			
S1	S. aureus	1 .1	0.92	NC010079 NC010063.1 NC012417.1	SRR022868	2. 9	101	38.1
S2	*E. coli*	21	19.4	NC_000913	SRR001655	4. 6	36	160.6
S3	*S. cerevisiae*	51	48.9	PRJNA128	SRX100885	12. 1	76	
S4	*C. elegance*	68	67.6		SRR065390	100.3	100	67
S5	H. Chrom. 14	36.2	35.1		NA	88.3	101	34

The S. cerevisiae reference genome of S3 is a concatenation of 16 chromosomes. Genome Length: Length of genomes without Ns. Number of Reads: Number of reads after all paired reads that contain Ns are removed and after trimming is done. Coverage: Number of Reads × Read Length/Genome Length. Error Rate: Mismatches/((Total Number of Reads - Unaligned Reads))* Read Length as defined in [119]

5.4.1.1 Evaluated Error Correction Methods

The methods evaluated for comparison to CECOND are the most recent error correction methods developed between 2012 and 2016. Some of the methods have since upgraded or modified the version of software used in the dissertation. The complete characteristics of the selected methods have previously been summarized in Table 3.1 of CHAPTER III. Versions of the methods are also presented in Table 5.3. Not all error correctors were used in the evaluation performed using experimental dataset. Having conducted an earlier analysis, the error correctors chosen for comparison with CECOND were 4 of the highest performing correctors evaluated and includes: BLESS, BFC, Bloocoo and Lighter. The only reason Musket did not make the cut is because of it takes a long time to run and consumes a lot of memory for large dataset.

Table 5.3 Version information of evaluated error correction methods and any associated tool used in the evaluation process

Error correction Method (Tool)	Version Used	Latest Version
CECOND	1.1	1.1
BFC	r181	r181
BLESS	0.23	1.01
Bloocoo	1.0.4-Linux	1.0.4-Linux
Lighter	1.1	1.1
Musket	1.1	1.1
Trowel	0.1.4.2	0.1.4.2

5.4.1.2 Computational Environment

All error corrections were performed on a server with a 64-bit Ubuntu 12.04 LTS Intel(R) Xeon(R) CPU with 16 nodes, 72 GB RAM and a core speed of 2.40GHz. The server also has MPI and OpenMp capabilities which is required by error correction methods like BLESS.

5.4.2 Evaluation Metrics

For a robust analysis, we also used two types of evaluation methods to validate the accuracy of CECOND namely; Validation through alignment and through genome assembly. For genome assembly, we used E. coli and timber rattlesnake data. For performance measure through alignment, we used the same metrics as in our prior studies [140]. The metrics are further explained here for clarity:

- Sensitivity: Sensitivity/true positive or recall rate which is a measure of correct identification of bases in the sequence i.e. if a base is correct or wrong, defined as:

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

- Specificity: Specificity / true negative rate which is a measure of correct identification of truly erroneous bases in a sequence defined as:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

- Accuracy: Accuracy in our context is a measure of the ability of a method to differentiate correct bases from erroneous ones and it is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Gain: is the ratio of the difference between pre-correction and post-correction error rate of a dataset to the pre-correction error rate of the dataset. Gain has a maximum value of 1 so the best methods should have values close to 1. Occasionally, a negative gain value is encountered indicating that a method introduced more errors during the error correction process. Gain is defined as:

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

This definitions are less stringent in comparison to definitions in [80], "any read containing errors was classified as TP provided at least one of its errors was detected and irrespective of whether they were accurately corrected or not." The above-mentioned metrics are important for the evaluation because they help us to:

1. Evaluate the ability of a method to detect an erroneous base
2. Identify methods that perform the wrong corrections despite discovering the erroneous bases in the read sequence.
3. Quantify the quality of error correction performed by each method
4. Determine if the amount of correction made is beneficial to the overall essence of error correction

5.4.3 Parameters

All evaluations were performed using the same number of threads or processes (12 for each case). This is to allow for accurate measurement of memory consumption and speed of error correction. For preprocessing, although CECOND can handle N's because it randomly converts all ambiguous bases to known bases, we still had to remove all N’s from the data before processing. Mainly, this removal of N’s is because of the inability of ECET [75], our evaluation toolkit to handle ambiguous bases. All other parameters were kept at the default settings of each individual method except for the k-mer parameter. Every tool requires this parameter and we had to derive a recommended k-mer value both from kmergenie [138] and looking at various histograms of k-mer values from jellyfish [172]. Plotting the histograms provides a means for checking peaks and determining regions of low and high k-mer coverage. Also, it allows us to determine what k-mer value will cover the genome. Several k-mer values were used for the evaluation and the results indicated which k-mer works best for an error corrector.

5.5 Results and Discussions

In furtherance of our earlier comparative analysis, which informed our decision to implement CECOND, performance evaluation of CECOND was in comparison to the same sets of error correction algorithms enumerated in CHAPTER III with versions used shown in Table 5.3. BLESS, BFC, Bloocoo, Lighter, Musket, Trowel and CECOND were used to correct simulated datasets of *E. coli, D. melanogaster* and H. chromosome 21. They were also used to correct experimental (real) datasets of *S. aureus, E. coli*, *S. cerevisiae, C. elegance, D. Melanogaster* and H. Chromosome 14. In this section, we present the results obtained

5.5.1 Evaluation of CECOND Using Synthetic Dataset

Following the same principles discussed in CHAPTER III, ART simulated reads consisting of 27 datasets (9 each) for E. coli, H. chromosome 21 and D. Melanogaster were generated. The data was de-novo corrected and evaluated using ECET by alignment with BWA according to the workflow shown in Figure 3.1. Due to the size of the dataset and desire to compare CECOND against dataset on which existing tools had the worst performance, selected subset of the data was used to evaluate CECOND in comparison to the existing analysis. The subset of data used are shown in Table 5.4. The results of performance comparison are presented in Table 5.5 based on true positives, true negatives, false negative, true negatives, gain, precision, recall and f-score. Selecting four important metrics, precision, recall, gain and f-score, for a clearer visualization of the performance, we illustrate the performance of CECOND in the form of a heatmap Figure 5.9.

Table 5.4 Subset of synthetic data used to evaluate performance of CECOND

Organism	Read Length (bp)	Genome Coverage
Escherichia coli (Ecoli)	50	20
Escherichia coli (Ecoli)	150	20
Escherichia coli (Ecoli)	50	80
Escherichia coli (Ecoli)	250	320
Human Chromosome 21 (Chr)	150	320
Drosophila melanogaster (Dme)	150	20
Drosophila melanogaster (Dme)	50	320

To evaluate performance of CECOND, it was compared to lighter, Bloocoo, Bless, BFC, Trowel, and Musket which are top five k-mer based methods. As mentioned earlier, same environment was used for the evaluation. Once the simulated reads were generated, preprocessing was done using ECET's preprocessing steps and kmergenie was used to select best recommended k-mer for each dataset. CECOND showed great performance in comparison to most of the correctors apart from coming second to BFC or third at the least in terms of performance. For genomes such as the human chromosome 21 data which has many repetitive regions and for larger sized genomes, CECOND generated the most accurate result with an f-score of 0.536 and 0.975 respectively. This is due to our implementation of sequence specific threshold instead of selection of a global threshold as implemented in several k-mer based methods. Often, most k-mer based method have problems with repetitive regions but not CECOND. Based on the results shown in Figure 5.9, we observe that the overall performance of CECOND is comparable to those of the top performers like BFC and BLESS

Table 5.5 Performance comparison of CECOND on simulated data with existing methods

Dataset	Method	TP	FP	FN	Recall	Gain	Precision	F-score
Ecoli 20X-L50	Lighter	524270	833102	2801156	0.158	-0.093	0.386	0.224
	Bloocoo	2437053	517156	888373	0.733	0.577	0.825	0.776
	BLESS	2370291	1412508	955135	0.713	0.288	0.627	0.667
	BFC	2729110	1578103	596316	0.821	0.346	0.634	0.715
	Trowel	159062	55354	3166364	0.048	0.031	0.742	0.090
	Musket	2411297	946045	914129	0.725	0.441	0.718	0.722
	CECOND	2458425	1167722	867001	0.502	0.270	0.684	0.579
Ecoli 20X- L150	Lighter	1833762	95317	221862	0.892	0.846	0.951	0.920
	Bloocoo	1791065	66800	264559	0.871	0.839	0.964	0.915
	BLESS	1987770	107360	67854	0.967	0.915	0.949	0.958
	BFC	2010306	109617	45318	0.978	0.925	0.948	0.963
	Trowel	206771	7639	1848853	0.101	0.097	0.964	0.182
	Musket	2023583	105268	32041	0.984	0.933	0.951	0.967
	CECOND	2019397	114261	36227	0.973	0.927	0.951	0.962
Ecoli 80X- L50	Lighter	1120010	166142	12188883	0.084	0.072	0.871	0.154
	Bloocoo	6049451	745885	7259442	0.455	0.399	0.890	0.602
	BLESS	8158797	3887266	5150096	0.613	0.321	0.677	0.644
	BFC	10306905	5536277	3001988	0.774	0.359	0.651	0.707
	Trowel	2956666	1033910	10352227	0.222	0.145	0.741	0.342
	Musket	4222269	964854	9086624	0.317	0.245	0.814	0.457
	CECOND	9257110	7092391	4051783	0.668	0.163	0.506	0.576
Ecoli 320X- L250	Lighter	467064	7640024	13752001	0.033	-0.505	0.058	0.042
	Bloocoo	12847399	105305	1371666	0.904	0.896	0.992	0.946
	BLESS	13869114	184149	349951	0.975	0.962	0.987	0.981
	BFC	14198589	169410	20476	0.999	0.987	0.988	0.993
	Trowel	3463360	45566	10755705	0.244	0.240	0.987	0.391
	Musket	13470253	67423	748812	0.947	0.943	0.995	0.971
	CECOND	12965445	104839	1253620	0.912	0.904	0.992	0.950
Chr 320X-L150	Lighter	95769002	34562312	188965339	0.336	0.215	0.735	0.462
	Bloocoo	89402571	46117811	195331770	0.314	0.152	0.660	0.426
	BLESS	95725880	6732168	189008461	0.336	0.313	0.934	0.495
	BFC	104590358	39031768	180143983	0.367	0.230	0.728	0.488
	Trowel	79032178	97825610	205702163	0.278	-0.066	0.447	0.342
	Musket	34494677	645021146	250239664	0.121	-2.144	0.051	0.072
	CECOND	**110086051**	16190168	174648290	0.387	0.330	0.872	0.536
Dme 20XL150	Lighter	49363299	2902247	8881569	0.848	0.798	0.945	0.893
	Bloocoo	14856304	136668296 1	43388564	0.255	-23.209	0.011	0.021
	BLESS	53060434	2325825	5184434	0.911	0.871	0.958	0.934
	BFC	56650226	2597501	1594642	0.973	0.928	0.956	0.964
	Trowel	8738	14491	58236130	0.000	0.000	0.376	0.000
	Musket	55657680	2937961	2587188	0.956	0.905	0.950	0.953
	CECOND	48611706	28968413	3500201	0.933	0.377	0.627	0.975
Dme 320XL50	Lighter	869521	44737030	127052556	0.007	-0.343	0.019	0.010
	Bloocoo	70628812	399843586	57293265	0.552	-2.574	0.150	0.236
	BLESS	99479127	830950017	28442950	0.778	-5.718	0.107	0.188
	BFC	94340796	707612613	33581281	0.738	-4.794	0.118	0.203
	Trowel	71539583	356478867	56382494	0.559	-2.227	0.167	0.257
	Musket	-	-	-	-	-	-	-
	CECOND	106903263	149354905	21018814	0.679	-4.240	0.121	0.206

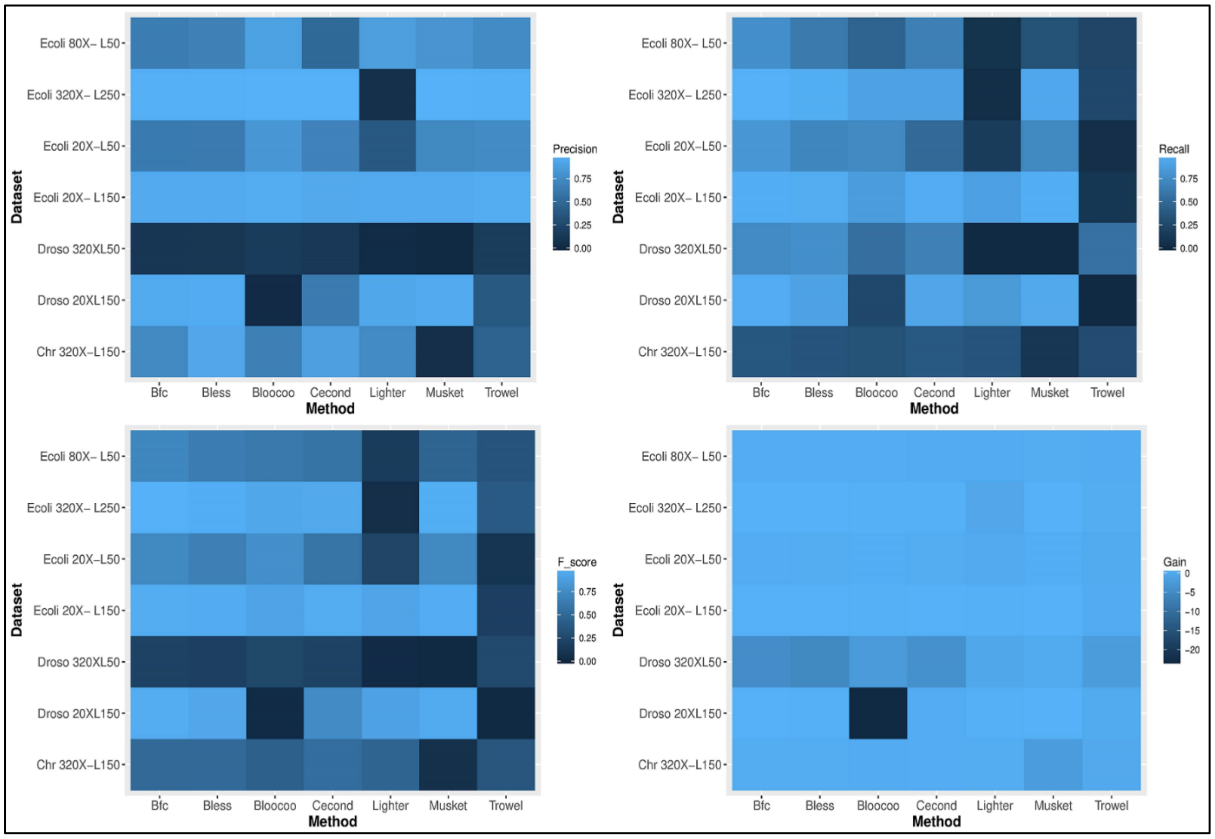


Figure 5.9 Heatmap illustrating performance of CECOND on simulated data in comparison with six k-mer based algorithms.

5.5.2 Evaluation of CECOND Using Experimental Dataset

To evaluate CECOND using experimental data, reference genome is instrumental for accurate evaluation. This is achieved by mapping the reads back to the reference genome and determining the performance from the counts of the mapped reads. A read will not map to the reference genome if it contains incorrect or erroneous bases. Mapping was done using BWA. The result of the performance is shown in Table 5.6.

Table 5.6 Alignment based evaluation result for experimental dataset

Data	Corrector	Accuracy (%)	Specificity (%)	Sensitivity (%)	Gain	Reads Mapped (%)
S1 S. aureus	Lighter	87.27	97.29	89.30	0.582	92.45
	BLESS	89.50	**97.59**	94.57	0.491	93.11
	BFC	**90.23**	89.06	92.50	**0.599**	**95.63**
	Bloocoo	87.72	85.93	**97.33**	0.564	91.07
	CECOND	89.45	92.50	93.15	0.491	89.56
S2 *E. coli*	Lighter	93.48	98.60	97.51	0.637	92.98
	BLESS	95.44	**99.72**	94.93	0.222	95.67
	BFC	95.90	99.37	96.64	0.496	93.68
	Bloocoo	**96.17**	99.10	97.79	**0.723**	**97.73**
	CECOND	95.83	99.47	**97.87**	0.655	95.91
S3 S. cerevisiae	Lighter	**96.40**	72.01	92.43	0.509	75.68
	BLESS	93.16	88.90	88.07	**0.873**	73.56
	BFC	95.52	87.56	88.42	0.678	**79.99**
	Bloocoo	90.72	**92.24**	91.01	0.701	79.03
	CECOND	92.34	73.67	90.70	0.682	74.45
S4 *C. elegance*	Lighter	89.71	97.55	87.77	0.572	81.74
	BLESS	92.33	**98.32**	92.30	0.523	80.10
	BFC	**92.78**	97.71	**92.33**	0.449	**89.23**
	Bloocoo	92.65	96.99	90.99	0.473	88.82
	CECOND	89.92	98.42	87.42	**0.581**	86.17
S5 H. Chromosome 14	Lighter	58.49	87.59	90.88	0.216	76.03
	BLESS	69.72	90.07	89.20	0.451	80.27
	BFC	**76.66**	85.42	**93.67**	0.503	82.11
	Bloocoo	70.57	**92.14**	90.80	0.550	78.28
	CECOND	75.77	91.92	92.73	**0.592**	**82.23**

Considering the performance results shown in Table 5.6, it is evident that BFC showed the overall best performance especially for accuracy and percent of mapped reads. CECOND performed well for S5 and had the best mapped rate at 82.23. It also showed better gain for S5 and S4 with a higher sensitivity for S2. Although the accuracy of CECOND is not as good as that of BFC or BLESS, it came a close second quite often and sometimes supersedes performance of lighter and BLESS (S5) or even Bloocoo (S1) which had a close performance to BFC. BLESS showed best performance in terms of specificity S1, S2 and S4 while lighter had the best performance for S3.

Mapping alone may not suffice for accurate evaluation. To ensure proper evaluation we went a step further to perform genome assembly. This is an important step because error correction affects genome assembly in crucial ways. To ensure that results of genome assembly are solely due to performance of error correction methods on the chosen dataset, we chose assemblers that do not have a built-in error correction algorithm or implement an external standalone error correction algorithm. This is unlike any other evaluations performed for other error correction methods

For this purpose, we choose SOAPdenovo – SOAP (Short Oligonucleotide Analysis Package) version 2.04 [173] with its error correction module turned off. SOAPdenovo also use iterative k-mer values, so we can adequately specify and compare assembly output of using several k-mer values. Specifically, we tried k-mers that are between 2 and 5 (inclusive) steps different from recommended best k-mer by kmergenie [138]. No gap closing was performed because we were only interested in capturing the values of the contiguous sequence generated based on error corrected data from the error correctors.

Table 5.7 Assembly based statistics for experimental datasets

Dataset	Corrector	N50	NG50	Edits/ 100kb	Misassemblies	Coverage
S1 *S. aureus*	Lighter	26947	26035	6.32	6	96.368
	BLESS	27440	27510	7.13	4	97.394
	BFC	27801	27786	6.68	7	96.450
	Bloocoo	26041	26042	5.49	9	97.225
	CECOND	27627	27629	6.01	13	97.107
S2 *E. coli*	Lighter	90001	90001	4.21	3	98.203
	BLESS	96410	96410	4.41	3	98.668
	BFC	96220	95636	5.02	4	98.711
	Bloocoo	98317	98317	7.32	1	98.541
	CECOND	98622	98622	4.11	2	98.720
S3 *S. cerevisiae*	Lighter	22675	22794	3.56	17	96.44
	BLESS	22983	23021	4.90	31	97.20
	BFC	23491	23491	7.45	9	96.40
	Bloocoo	23510	23510	4.47	21	95.39
	CECOND	22910	23001	4.55	45	93.90
S4 *C. elegance*	Lighter	18056	17947	26.55	522	96.201
	BLESS	18923	18973	27.33	462	94.871
	BFC	19435	20167	29.10	473	95.407
	Bloocoo	17422	17611	29.62	417	95.518
	CECOND	18713	18903	27.30	485	95.420
S5 H. Chromosome 14	Lighter	5756	4218	126.7	569	77.218
	BLESS	5862	4240	132.3	607	78.436
	BFC	5894	4220	128.2	720	78.727
	Bloocoo	5730	4119	133.7	654	79.155
	CECOND	5823	4101	127.1	582	79.224

5.5.3 Computational Resource Consumption

For measurement of the computational resource consumption of CECOND, we used the time command and memusage like in our previous analysis in CHAPTER III. The result of the comparison is shown in Table 5.8 for simulated dataset and Table 5.9 for experimental dataset. The memory footprint of CECOND is comparable with those of existing methods. It is next to BLESS and Lighter reasons been that both methods allocate a constant amount of memory. CECOND allocates memory dynamically like BFC and Bloocoo. With an average memory usage of 3.30 and 3.81, its footprint is lower than BFC (7.48 and 4.82) and Bloocoo (5.43 and 4.08) for both simulated and experimental datasets respectively. Variation in memory usage is also respectably comparable with those of existing methods as observed from the standard deviations presented in the table.

Speed wise, the error correction running time by CECOND is acceptable given that it is a novel implementation and can be improved. Generally, the speed of CECOND scales with the amount of memory allocated. For simulated data, its runtime for correction is on par with the rest of the methods. Though it took longer to correct Chr21 and Dme at 320 times coverage, it still took a shorter time (0.44) in comparison with lighter, bless and Bloocoo. Only BFC surpassed its speed. On the experimental data, it was bit grim for CECOND in terms of speed. With an average correction time of 0.89, it runs at about 65% the speed of the tools it is been compared with for real dataset. More test and analysis is required to determine areas of potential bottleneck for the speed to improve. Overall, its performance computational wise is good.

Table 5.8 Comparison of computational resource consumption of error correctors for simulated dataset memory(GB) and time(hrs)

Error Corrector	**BFC**		**Bless**		**Bloocoo**		**Lighter**		**CECOND**	
Measurement	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time
Mean	7.48	2.41	0.02	4.11	5.43	5.47	1.48	1.34	3.30	2.10
Standard deviation	6.17	5.42	0.00	7.41	1.69	9.17	0.61	2.12	2.60	3.40
Eco_20X_L50	2.14	0.01	0.02	0.01	4.11	0.01	0.35	0.00	0.31	0.02
Eco_20X_ L150	2.87	0.04	0.02	0.01	7.44	0.02	1.27	0.01	1.42	0.03
Eco_80X_ L50	3.99	0.22	0.02	0.33	4.36	0.02	1.47	0.02	2.33	0.15
Eco_320X_ L250	15.94	14.64	0.02	19.92	8.24	22.21	2.22	5.42	6.40	7.83
Chr21_320X_L150	15.94	1.61	0.02	7.00	5.01	14.81	1.90	3.06	5.11	6.22
Dme_20X_L150	2.78	0.10	0.02	0.02	4.33	0.02	1.27	0.01	1.11	0.03
Dme_320_XL50	8.71	0.22	0.02	1.48	4.49	1.21	1.90	0.84	6.40	0.44

Table 5.9 Comparison of computational resource consumption of error correctors for experimental dataset

Error Corrector	**Bfc**		**Bless**		**Bloocoo**		**Lighter**		**CECOND**	
Measurement	Memory	Time	Memory	Time	Memory	Time	Memory	Time	Memory	Time
Mean	4.82	0.60	0.02	0.69	5.08	0.56	1.23	0.54	3.81	0.89
Standard deviation	1.79	0.57	0.00	0.74	1.82	0.38	0.55	0.75	2.24	0.42
S1	3.22	0.22	0.02	0.24	4.15	0.29	0.82	0.15	2.17	0.40
S2	3.45	0.13	0.02	0.17	4.07	0.32	0.94	0.14	1.52	1.61
S3	5.61	0.49	0.02	0.27	4.28	0.51	1.18	0.21	3.10	0.82
S4	4.29	0.62	0.02	0.87	4.57	0.46	1.02	0.31	5.47	1.22
S5	7.55	1.56	0.02	1.92	8.32	1.23	2.19	1.87	6.79	1.41

5.6 Scalability

CECOND was run multiple times for *Escherichia coli* dataset S2 to observe its scalability. Since multithreading was implemented in CECOND, there was a need to measure the speed of the error correction process using different number of threads. The evaluation was performed by running error correction with threads of 1, 2, 4, 6, 10, 12, 14 and 16, which is the maximum for the system used in our test evaluation. The result is illustrated in Figure 5.10. From this figure, we observe an almost linear speedup when the number of threads used was varied from 1 to 16. Due to the implementation of CECOND, there was a variation in the amount of memory consumption given different number of threads. The memory bottleneck stems from the way supplied data is been read by CECOND which affects the runtime. The amount of data processed at each point increases with increased number of cores thereby decreasing the run time.

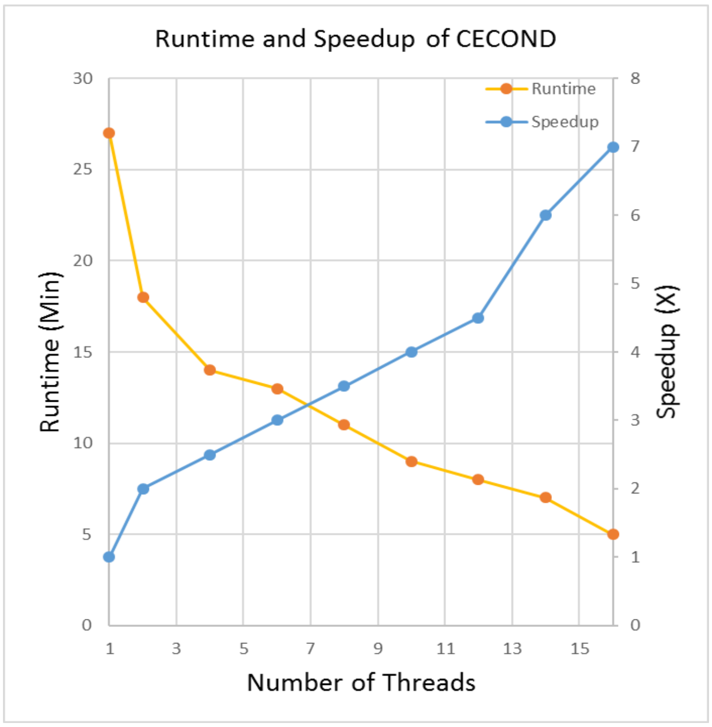


Figure 5.10 Runtime and Speedup of CECOND for E. coli data

5.7 Recommendations to Users

To facilitate efficient error correction by CECOND, we advise that data Illumina data should be used. It is currently targeted for correcting such data and all evaluations performed in this chapter were Illumina specific. Users should ensure that their data is preprocessed through quality and adapter trimming. CECOND will not accept data (fastq file) that contains reads with average quality score less than 20.

k-mer values should also be optimal. We recommend using an existing k-mer value determination method like kmergenie to evaluate the best k value for each dataset. The general rule of thumb is to set the k-mer value to about two third of the length of the sequence. E.g. for a read length of 56, the recommended k-mer will range between 19 and 36. The k-mer value is integral to the number of errors that can be corrected and this is true for all k-mer based methods. If runtime or memory is not an issue, we recommend specifying a range of k-mer values so that CECOND will recursively run the same dataset but with different k-mer values. Be warned, this may take quite some time depending on the size of the dataset and the error rate of the sequence.

Typically, we recommend a computing environment with at least 4Gb of memory depending on the dataset. For large genomes, if the coverage is high enough e.g. 60X and above, we recommend large k-mers and a 4GB memory will allow CECOND to run smoothly. Large k-mers allows errors effective correction of errors that occur in repetitive regions thereby improving the accuracy of CECOND.

5.8 Limitations

1. The current implementation of CECOND is its first and development is still ongoing. Some more fine tuning is required to allow for its full capabilities to be achieved.
2. The amount of memory consumed is excellent for small genomes but not so for larger genomes. CECOND requires a more efficient way of reading the data from large files to further reduce the amount of memory used. This is because it allocates memory progressively as data is being read and the hash table gets expanded.
3. CECOND is currently unable to handle reads containing indel errors. There is a need for further analysis of the threshold method implemented in CECOND to allow for indel correction.
4. Several parameters are required to be given by the user. Upon further analysis, the number of parameters required by CECOND may be reduced.
5. Finally, it is worth considering the use of multiple k-mers iteratively during the error correction process. The method has been implemented in some genome assembly algorithms and can also be implemented in error correction algorithms. Making use of several k-mers gives an assurance of better error correction.

5.9 Software Information

The current version of CECOND, which is the first release is version 0.15. Testing was conducted on GNU/Linux Ubuntu 14.04.5 with GCC 4.8.5 and requires pthread. Included in the folder is a readme file showing details of CECOND including installation instructions, usage instructions and sample test data. The software is available under the GNU (General Public License) version 3.0 (GPLv3) without any restrictions to use or modify by non-academics. CECOND can be freely downloaded and extracted from http://pinfish.cs.usm.edu/cecond.

5.10 Contributions of this work

This chapter elaborates a novel error correction method that implements cuckoo filter as its underlying data structure. The efficiency of CECOND relies on the efficiency of the cuckoo filter for error correction in a way, similar to how lighter [124] relies on bloom filter. The contributions of this work include:

1. Combines Cuckoo filter and hash tables in a memory efficient way to reduce the amount of computational resources consumed during the error correction process.
2. Implements a different data structure that has not been tested on NGS data prior to this implementation. It produced results comparable to those of several existing k-mer based error correction methods.
3. The method of threshold selection ensures that errors that occur in repetitive regions are corrected as opposed to several k-mer based error correction methods which use only one global threshold for correcting the entire dataset

5.11 Chapter Summary

In this chapter, we discussed a novel error correction algorithm, CECOND, which is based on cuckoo filter data structure. The method of error correction and threshold selection was discussed. Upon development, the method was tested on both experimental and simulated data, compared against state of the art k-mer based methods and evaluation results was presented. The results showed comparable accuracy with existing methods even though it is the first implementation of this data structure for set membership test (other methods used bloom filter in the past). Furthermore, the importance of CECOND for the correction of repetitive regions of a genome was highlighted in the result. In addition, despite the good qualities of CECOND, its limitations were also highlighted. The limitations may act as a guide for future further improvements.

CHAPTER VI TIMBER RATTLESNAKE GENOME ASSEMBLY

Several tetrapod vertebrates, specifically the serpent suborder, have been studied and assembled with annotated genomes e.g. the Burmese python and king cobra genomes. Despite timber rattlesnake's wide habitat in the Americas and their diversity, little effort has been made to study them until recently. Timber rattlesnakes (*Crotalus horridus*) which we will refer to as TR, poses important characteristics that makes them excellent candidates for novel research questions. These snakes exhibit a wide selection of important phenotypic characteristics that may have implications for humans. Their ability to withstand long starvation bouts, hibernate, slow down their metabolism, decrease or increase their anatomical features like the hearts and lungs to facilitate survival and the wide variety of species available are some of the reasons for their study. Considering these important characteristics, it is a surprise that they have not been sequenced and assembled until now. Here we implement our error correction method, CECOND together with an existing error corrector, BFC, to ease assembly of TR genome.

6.1 Overview

Improvement in routine generation of sequence data due to the availability of better sequencing technologies, as discussed in CHAPTER I, has led to increased amount of de-novo based assembly being performed. While many researchers have been involved in the genome assembly of various organisms, not much attention was given to vertebrates until recently. The Genome 10k project [174] whose goal is to sequence 10,000 vertebrate genomes was one of the projects that revolutionized sequencing of vertebrates.

The difficulty posed by de novo assembly, notwithstanding the improvement in genome sequencing data generation, led to competitive efforts like the Assemblathon II project [175]. The outcome of such projects indicated that assembly quality is dependent on several factors including the assembler design, parameters, complexity of the sequenced organism, and data quality. Assemblers are generally grouped into 3 major categories: De-Bruijn, Overlap and hybrid based. Vertebrate organisms can be very complex making it difficult to assemble due to their repeat contents. Timber rattlesnakes is one of such organisms which have undergone phenotypic and morphological transformations. The focus of this chapter is to understand how error correction will affect genome assembly of such a previously unassembled organism. In this chapter, First, a de-novo genome assembly of the TR was performed using existing dataset and compared with other available assemblies from the same dataset. Secondly, additional Illumina based data was generated. Before error correction, all data were checked for adapters and low quality bases which were subsequently removed through adapter and base quality trimming [176]–[178]. Finally, the data was error corrected using CECOND and BFC. Error corrected TR genomic data from BFC and CECOND was used in the genome assembly process discussed here. The results indicate assembly improvement because of error correction. This chapter discusses the process that led to assembly of the TR genome.

6.2 Data and Materials

The initial raw sequence data consisted of 454 reads, Illumina paired end and mate-pair reads. To achieve a better assembly, more Illumina Miseq and Hiseq data was sequenced. Table 6.1 shows the source of the sequencing sample. The juvenile was born in captivity and was properly sacrificed for RNA isolation according to standards because there was no need for it. The initial raw sequence data generated after sequencing and the sequencing platforms are shown in Table 6.2. For a more expansive analysis, more data was sequenced. The additional data is shown in Table 6.3

Table 6.1 Source of sample extracted for sequencing the timber rattlesnake genome

Data type	**Isolation source**	**Sex**	**Development stage**
Genomic	Blood	Female	Adult
RNA-seq	Muscle, blood, heart, head, digestive tract, mixed internal organs	Male	Juvenile

Table 6.2 Raw sequence fastq dataset for initial Assembly

Reads	Source	Library	Number of files	Size (gb)	Coverage
454	gDNA	Single read	13	23.540	11.8X
Illumina PE	gDNA	100bp on 179bp library	4	106.235	52.4X
Illumina MP	gDNA	100bp on 6.6kbp library	2	44.937	21.5X
Illumina PE	Mixed tissue RNA	265bp (including adapters 130bp)	2	10.534	5.3X
		Total		181.712	90.9X

While the Roche 454 reads are single libraries, all Illumina data are paired libraries where PE – Paired-end and MP -Mate-pair. The total coverage shown is just to show the coverage of the dataset not the coverage of the genome. The coverage of the genome are the individual values shown. gDNA refers to genomic DNA. The file sizes are in giga bases

Table 6.3 Additional Illumina Miseq and Hiseq data with SRA accession number

SRA Accession No.	Technology (Illumina)	%GC	Insert Size	Read Length (bp)	Total nt Sequences	File Size (Gb)
SRR3185239	Miseq	40	350	2 X 300	11847896	15.2
SRR3185241			350	2 X 300	12211478	15.6
SRR3185252			550	2 X 300	13084150	16.8
SRR3185265			550	2 X 300	13289092	17.0
SRR3185268	Hiseq	39	350	2 X 100	80263916	38.8
SRR3185269			350	2 X 100	70880665	34.3
SRR3185271			350	2 X 100	75011492	36.3
SRR3185272			550	2 X 100	98275350	47.5
SRR3185274			550	2 X 100	94569070	45.7
SRR3185275			550	2 X 100	93517900	45.2
			Total File Size			312.4

6.3 Computational Environment

All genomic assembly runs were performed on USM's School of Computing pinfish server running on a CentOS 64-bit Intel(R) Xeon(R) CPU E5630@ 2.53 GHz machine with 16 processors, 296 GB RAM and a total storage of 8 TB. The trinity assembly for RNA-seq data was run on USM's BigCat server with a 64-bit Ubuntu 12.04 LTS Intel(R) Xeon(R) CPU with 16 nodes, 72 GB RAM and a core speed of 2.40GHz.

6.4 Methodology

The Roche 454 dataset and gDNA Illumina paired-end dataset were used for the genome assembly while the Illumina mate pair information was used to extend the assembly. To assemble the genome, raw sequence data was first preprocessed to remove contaminants and errors. The pyrosequencing 454 data was in a binary SFF (Standard Flowgram Format) file and the sequences had to be extracted and subsequently converted to fastq format using Sffinfo of mothur [61]. The Illumina and 454 datasets were first preprocessed and quality checked after which error correction was performed.

6.4.1 Data Pre-processing

To prepare the data for genome assembly, contaminants like adapter read through and low quality bases were removed through trimming using trimmomatic [176]. Erroneous data can also lead to slow assembler run, RAM consumption and poor or misconstrued results. Pre-processing eliminates construction of suboptimal paths during genome assembly, reduce sequence volume thereby enabling easier processing. To ensure that the right number of bases and qualities are trimmed, there was a need to examine the quality of the raw sequence files. For example, a quality score of 13 is equivalent to a 5%

error. Such values can be trimmed based on base quality, using a sliding window or percentage of good quality sequence per window. A GUI based QC (quality control) tool fastqc [179] was used. A subset of observation from fastqc based on the unprocessed timber rattlesnake genome is shown in Figure 6.1. Low quality scores are observed at the 3' end with hints of adapter sequences in position 1 to 6 at the 5' end. The main idea was to check for and remove over represented sequences, low quality bases and adapter sequences. Further processing was performed using fastx toolkit [178] and NGS QC Toolkit [177] both of which does quality trimming as well as formatting fastq files to remove single unpaired sequences. The before and after preprocessing statistics for the initial sequencing data is shown in Figure 6.2 (a) and (b). The impact of trimming is clearly visible. A 6% increase in final contribution to the data is observed for the Illumina PE data and an 8% decrease for 454 data.

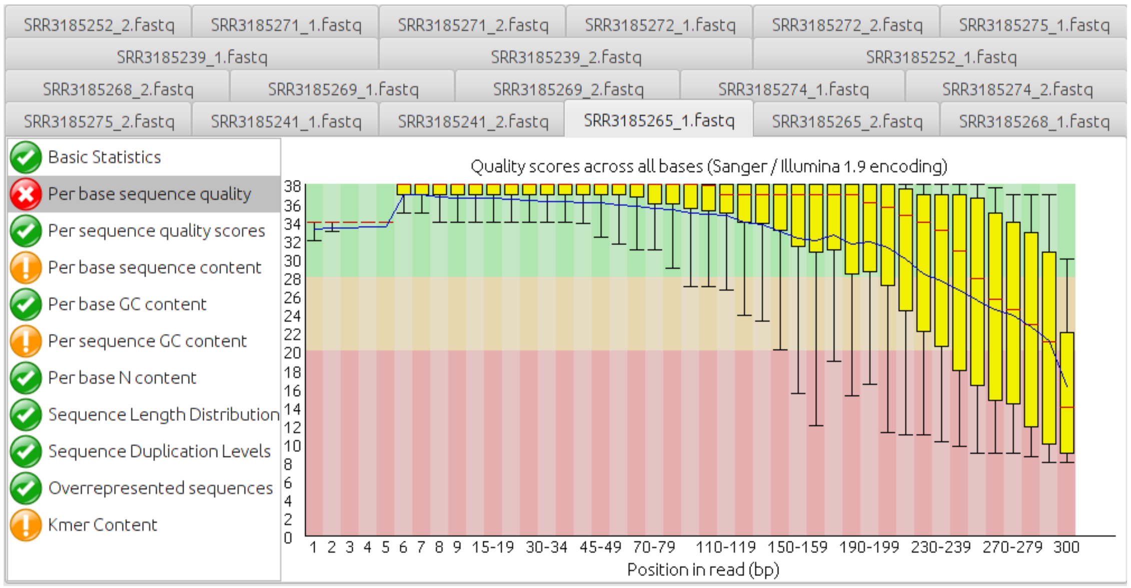
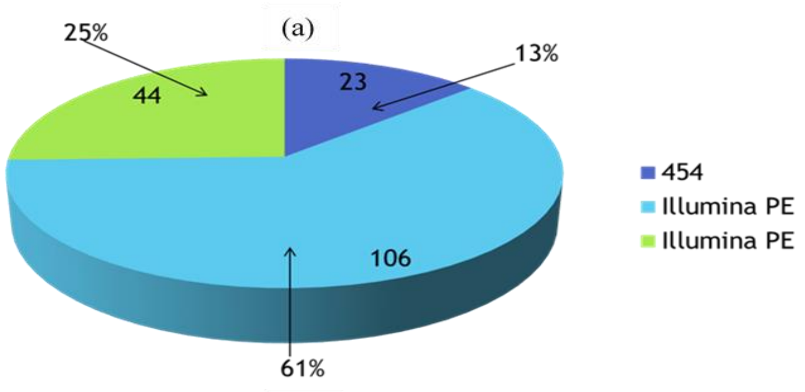


Figure 6.1 Fastqc representation of per sequence quality score of SRR3185265_1. fastq



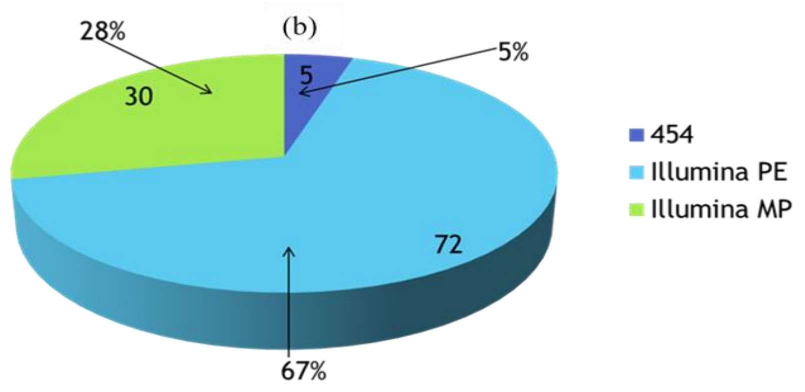


Figure 6.2 Size of timber rattlesnake data (a) before trimming (b) after trimming

6.4.2 Correcting Timber Rattlesnake Data

After preprocessing steps were completed. The processed data was used as input for both CECOND and BFC. The choice of BFC for comparison with CECOND, given the experimental TR data, is because BFC was determined to have the best performance overall after evaluations conducted in both CHAPTER III and CHAPTER IV. Several steps were used to determine how to proceed with error correction and are briefly discussed in this section

6.4.2.1 K-mer Selection

CECOND, which is based on cuckoo filter data structure as described in CHAPTER V, and BFC, were used to correct the TR dataset. Both CECOND and BFC requires k-mer as one of the main inputs used in their error correction process. Kmergenie [138] was used to determine the best K for the given data. Kmergenie produced a recommended k of 35 with a total of 1,087,547,810 genomic k-mers. This value is close to the genome size of TR genome. This can be improved after error correction and subsequent genome assembly steps are performed. For a more robust analysis to validate kmergenie output, we decided to experiment with several values of k centered around the recommended k of 35 by kmergenie. We increased and decreased the k value by multiples of 4. The final values of k used for the experiment are: 23, 27,31, 35, 39, 43 and 47. To evaluate the k-mer values shown, DSK [128], a k-mer counting method was used. DSK is like Jellyfish and provides several valuable information based on the value of k provided to it as input. It also gives an estimated expected genome size based on the k value. The results of DSK run on the TR data from multiple values of k are shown in Table 6.4.

Table 6.4 Output of DSK run on multiple values of k for evaluation of kmergenie recommended k of 35

K value	**27**	**31**	**35**	**39**	**43**
Number of k-mers	**6,681,306,980**	**6583769660**	**6,486,232,340**	**6,388,695,020**	**6,291,157,700**
Valid K-mers	4421045121	4275877951	4141414906	4014962021	3890894050
Invalid K-mers_	2355091390	2318320169	2270844823	2383852451	2410228598
Distinct k-mers	**1768216733**	1830169426	1876875964	1912113725	1936457140
Solid k-mers	472811440	452252452	491813252	430824463	4710228598
Weak k-mers	1357357986	1424623512	1276403481	1481289262	1227909980

The results confirm that the k value generated by kmergenie is optimal given that there is a decrease in the number of solid k- for every subsequent k-mers with values 4 places below or above the k value of 35 chosen by kmergenie as optimal. Although the number of distinct k-mers increases steadily as the value of k increases, the value of solid k-mers was highest at k = 35. The total number of k-mers observed decreases as the value of k increases. This is totally normal because the smaller the value of k, the more k-mers are generated. The value of weak k-mers is also minimal at k = 35 although it is higher than at k = 43. Having a lower value implies lesser error complexity for the error correctors. After error correction, a decrease in the value of weak k-mers present in the read is expected at the given k value of 35.

6.4.2.2 Error Correction

Once it is determined that the value of k generated by kmergenie from the process in 6.4.2.1, is valid, we proceed to error correction of the data. An inherent difficulty in most error correction method is the time it takes to process the large amount of data generated from NGS sequencing technologies especially, from complex genomes like those of TR, been considered in this work. This was not a problem with both BFC and CECOND due to their design. The data was processed using both methods and evaluated. The results of evaluation of the corrected data were compared and shown in Table 6.5TABLE 4. Due to the lack of a well annotated reference genome assembly for TR, after error correction, the error corrected reads were mapped back to the genome of the Burmese python which is a closely related specie.

This helps to determine the amount of error free reads from the counts of the mapped reads. A read will not map to the reference genome if it contains incorrect or erroneous bases. Mapping was done using BWA. The same process used in measuring performance of error correction method in 5.5.2 was used for this evaluation.

Table 6.5 Evaluation of CECOND and BFC performance on TR data with Burmese Python as the reference genome

Metrics (%)	**Error Corrector**	
	CECOND	**BFC**
Accuracy	67.54	72.16
Specificity	83.11	81.73
Sensitivity	77.32	80.20
Gain	0.401	0.449
Reads Mapped	77.98	78.34

The percentage of reads mapping to the Burmese python reference was not very high but it gives us an idea of how the correctors performed. Given the limited information we have (no reference genome for timber rattlesnake), using a close relative was the best method in our case. Although BFC still performed better than CECOND, the ability of CECOND to detect the errors present in the reads is much higher. This is because of the way CECOND builds local thresholds for identifying errors in each read. Apart from BFC being more accurate than CECOND, every other metrics generated are comparable with only a maximum difference of about 0.5%. Based on the result, corrected reads from any of both methods can be used as input to MaSuRCA for generating the de-novo assembly. The decision was made to use both results from the error correction as input to the assembler. The result from CECOND was used as input to the assembler while the result from BFC was used to refine the assembly. This was achieved by mapping the reads to the assembly during the extension process. The genome assembly process is discussed in the next sections.

6.4.3 De-novo Genome Assembly Process

After preprocessing the data to remove contaminants and errors, de-novo genome assembly was performed. Genome assembly, which is the process of determining the ordering of the bases and stitching them back to form contiguous sequence has been elaborated in CHAPTER II. Several genome assembly tools are available. For genomic assembly, the whole genome MASURCA (Maryland Super Read Cabog Assembler) [152] was used. It was chosen because of its ability to work with reads of variable length (due to trimming) and reads from multiple sequencing platforms – it is an efficient hybrid approach that combines de Bruijn graph and overlap-based assembly methods. QUAST (Quality Assessment Tool for Genome Assemblies) [180] was used to compare MaSuRCA assembly (using default values) with another efficient assembler, IDBA-UD [181] before its selection as an ideal assembler for our genome assembly. See Figure B.2.

6.4.4 De-novo Genome Assembly and Comparative Result

With timber rattlesnakes not having been assembled before, there was no draft genome available on genomic databases. De-novo based methods should be used. We stated (see 5.4.2) the reasons why MaSuRCA [146] was chosen as the ideal assembler in our case. A prior de-novo assembly had been generated using velvet [178]. The assembly used the initial data shown in Table 5.3. To further analyze the data to measure effect of assembler program on a genome assembly, we used the same data to generate an assembly by MASURCA. Before assembly, kmergenie [138] was used to determine the best recommended k based on our data. The recommended k was used as input for assembly. The workflow used in generating the assembly is shown in Figure 6.3

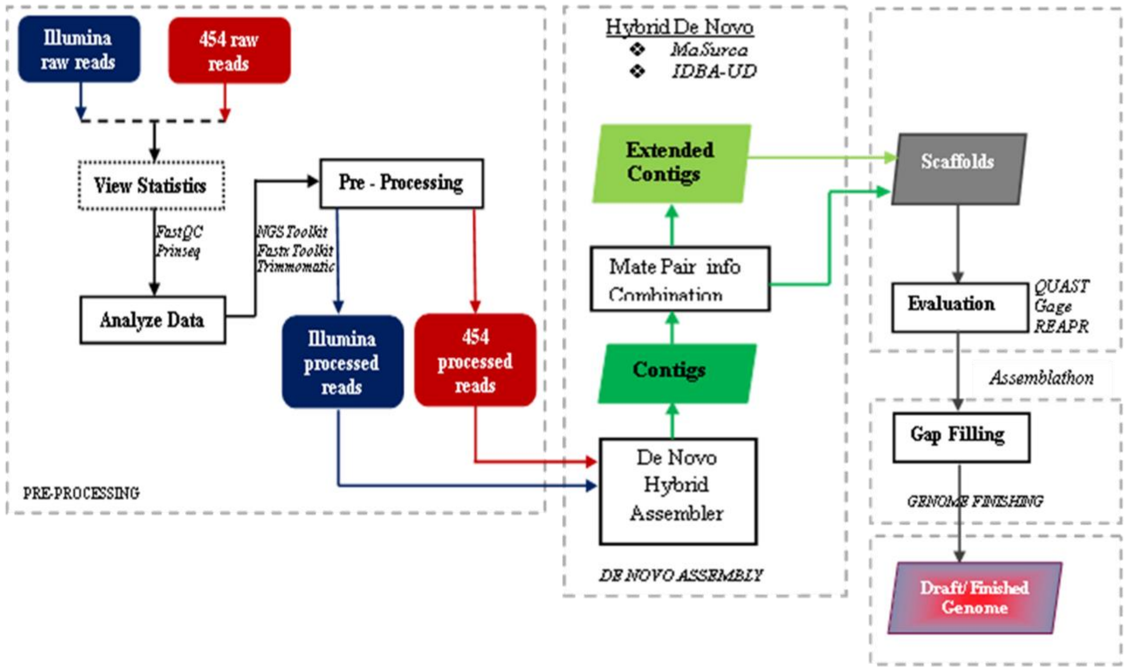


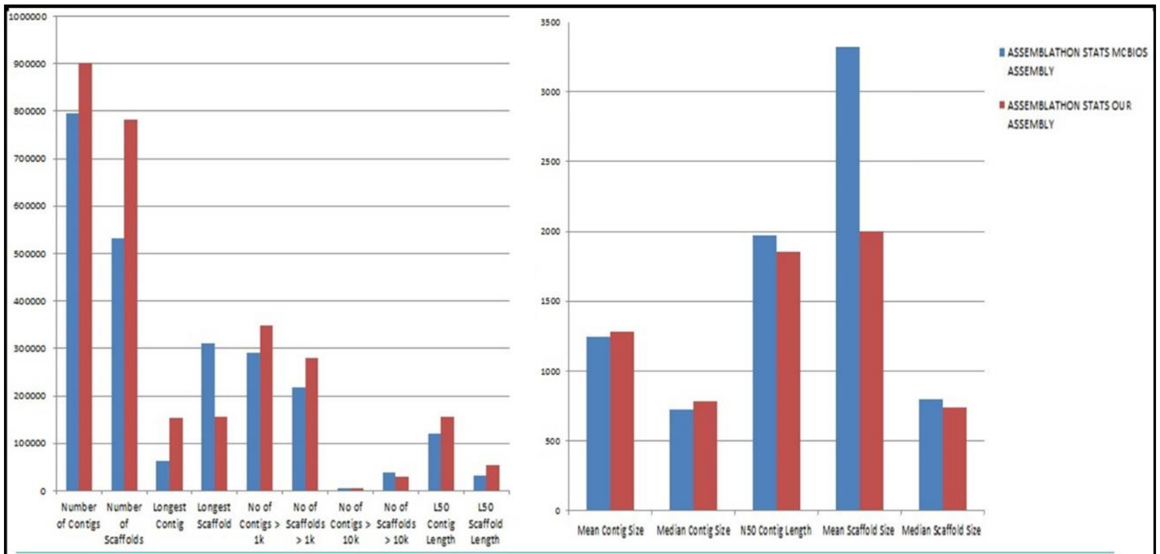
Figure 6.3 Timber rattlesnake assembly workflow with Assemblathon [175] used for assembly comparison

Genome assembly was performed based on initial available sequence data, Table 5.3, and compared with an existing timber rattlesnake genome using the same dataset. The assembly result showed higher quality to existing velvet based assembly Figure 6.4

but was still well short of the known genome size for timber rattlesnake. The generated scaffold lengths were also well shorter than the values observed for both the Burmese python [182] and King cobra [183] genome. For these reasons, additional data was sequenced. The final assembly was performed using a combination of both datasets in Table 6.2 and Table 6.3. which had been error corrected in the prior section using combined CECOND and BFC corrected data. The result of the final assembly is shown in Figure 6.6 for both error corrected and non-corrected datasets.

An N50 contig size of N implies 50% of the assembled bases are contained in contigs of length N or larger. N50 sizes are often used as a measure of assembly quality because they capture how much of the genome is covered by relatively large contigs [184] The N50 is like a mean or median, but with greater weight given to the longer contigs. N50 can be longer when the measurement falls within an area with longer contigs than others while it could be shorter if it falls between the region of the assembly with shorter contigs hence not a very accurate measurement of the quality of the assembly but will suffice for a de novo assembly since there is no reference genome. The initial assemblies are comparable with better statistics in some respects. Overall statistics of our assembly is comparable and slightly better with longer MaSuRCA based contigs. Even though our longest scaffold was relatively shorter, we can see that the number of scaffolds greater than 1k NT (Nucleotide) is much higher and those greater than 10k is comparable. Our L50 contig and scaffold counts are also much better. The L50 measure is the number of scaffolds/contigs that are greater than, or equal to, the N50 length [175].

Having a higher L50 count demonstrates that the MaSuRCA assembly is of a higher quality than the velvet based assembly.



Assembly	N50 Contig Length (Kbp)	N50 Scaffold Length (Kbp)
Velvet (Rattlesnake)	1.853	13.4
MaSuRCA (Rattlesnake)	1.969	7.4
Burmese python (published)	4.097	183

Figure 6.4 Comparison of Velvet based and MaSuRCA based assemblies. Assemblathon statistics shows significantly low values for both contig and scaffold N50 values in comparison to values from the Burmese python genome assembly generated in [182]

After performing error correction on the data, de-novo genome assembly was performed. The existing assembly with results shown in Figure 6.4 was combined with the new assembly generated and extended into scaffolds using the Illumina mate-pair data. To merge both assembly, SSPACE (SSAKE-based Scaffolding of Pre-Assembled Contigs after Extension) [185]. Combining the assembly in this way helps create longer contiguous sequence. The result of the final assembly generated are shown in Figure 6.5 and Figure 6.6. The results were compared against an existing draft genome deposited on

NCBI database by the MCBIOS (Mid-South Computational Biology and Bioinformatics Society) community effort. Figure 6.7 compares the total genome size of the new assembly with that of the existing and uncorrected data assembly. The idea is for this new assembly to be evaluated and used as an update to the version on NCBI.

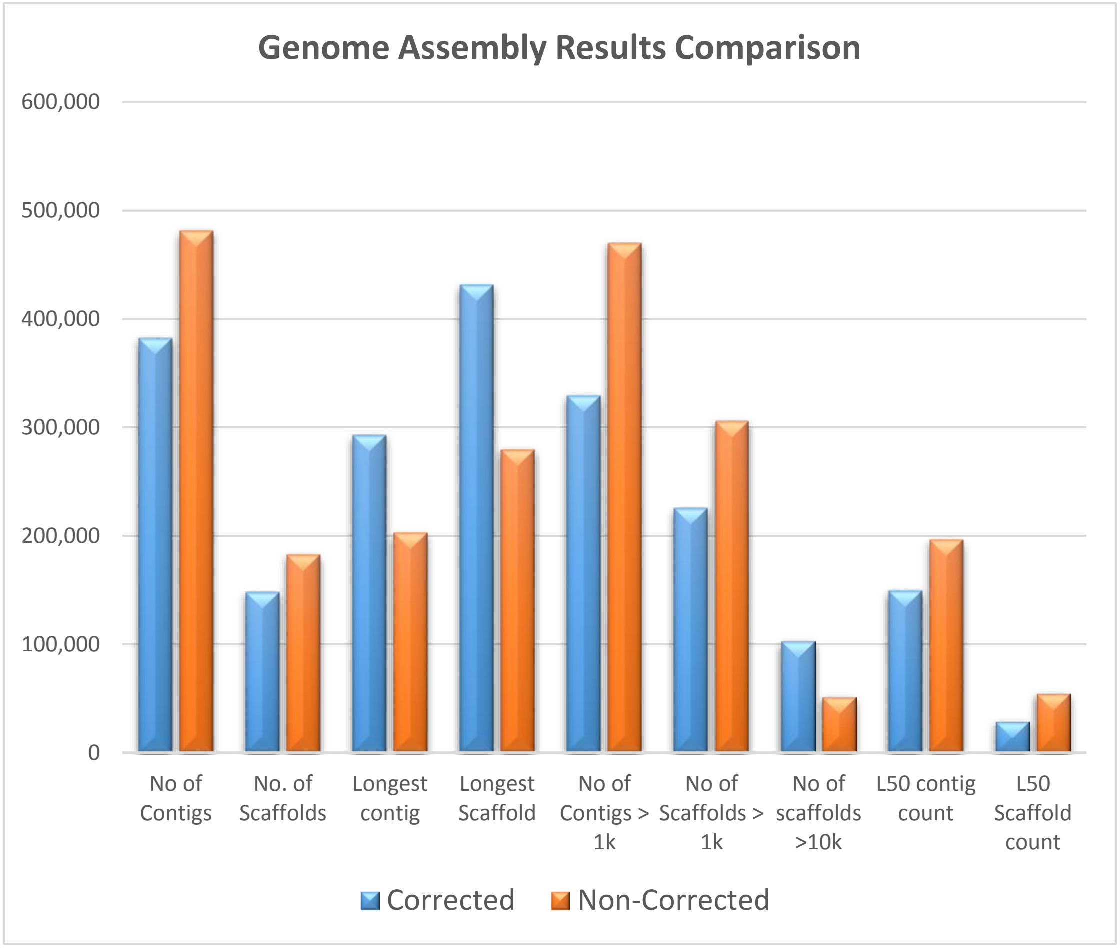


Figure 6.5 Assemblathon assembly evaluation for corrected and non-corrected data

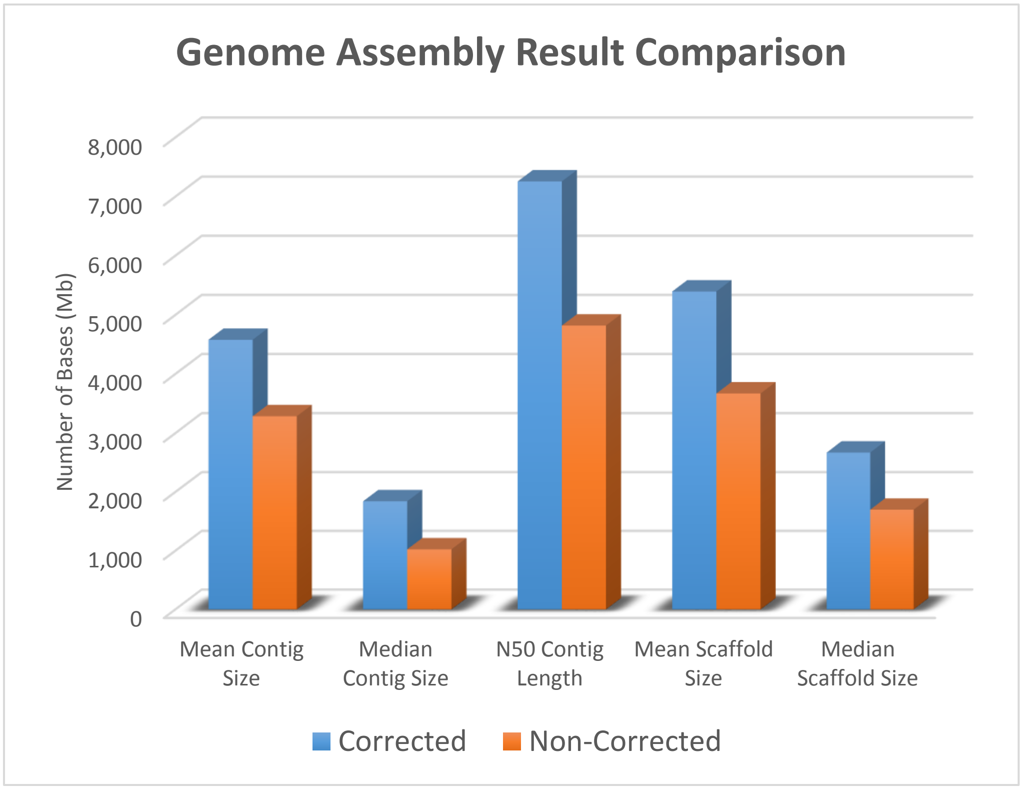


Figure 6.6 Contig and scaffold size comparison for corrected and non-corrected data

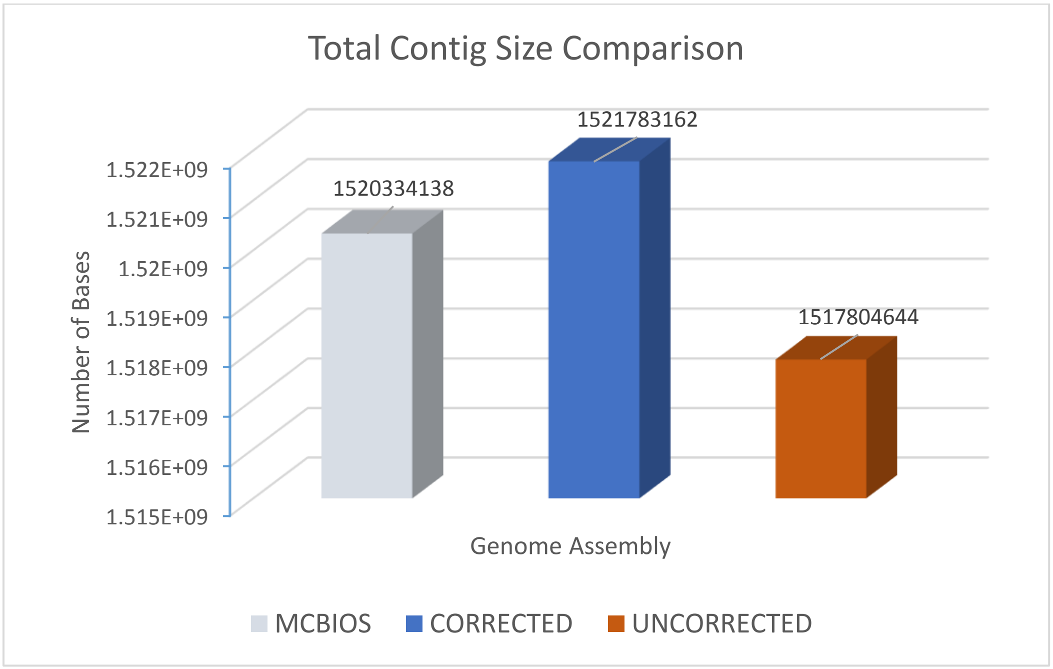


Figure 6.7 Comparison of total contig size of our corrected assembly against assembly of uncorrected sequence and MCBIOS assembly deposited on NCBI database

6.5 Discussions and Conclusions

With the critical nature of genome assembly for various genomic downstream analysis, it is necessary that the correct ordering of the genomic sequence is attained. Having a properly sequenced data, engaging in proper pre-processing steps and performing qualitative analysis is relevant for genome assembly. From our results, error correction has shown to improve the quality of the assembly especially in terms of contiguous sequence length and total contig size (length) or genome size. With an initial size of 1,517,804,644 total number of bases for uncorrected data, error correction lead to an increase of 3,978,518 contiguous bases for the assembly. This is an extremely significant number given that longer contiguous bases are desired in genome assembly. Without any proof to determine if error correction was performed or if a combination of existing data and newly generated data was used for the deposited assembly on NCBI, we compared the resulting assembly from our error corrected data. The assembly also showed an improvement of over 1.4 million bases in the contiguous assembly. This increase in the total contig size is important for conclusions that can be drawn from such assembly data.

We conclude that as far as genome assembly and analysis is concerned, error correction is significant for qualitative analysis. Determining the type of error correction method to use is also important because methods that may work on genomic data almost always does not work well with RNA sequence data. Improvement in the sequencing assembly is largely due to the error correction made as observed in the difference between the corrected and uncorrected genome assembly.

6.6 Chapter Summary

In this chapter, we:

- Discussed timber rattlesnake as a novel model organism for several studies
- Applied error correction to Illumina based genomic data of the timber rattlesnake to investigate the effectiveness of error correctors for real novel datasets and analyze improvements that can be made upon implementation

We observed that

- Error correction is an important facet of every genome assembly project which is buttressed by the fact that almost all genome assemblers rely on either internal or external error correction method to resolve ambiguities in their graph during contiguous sequence reconstruction.
- The implementation of error correction to the rattlesnake data generated a better genome assembly than that of an uncorrected sequence data. This is indicative of how effective error correction can be in improving genome assembly.

We conclude that

- Error correction is an important step that should never be overlooked during a genome assembly process
- Error correction can adversely affect the quality of a final assembly and the method for error correction should be chosen with care to prevent over correction which may lead to the introduction of more errors

CHAPTER VII – CONCLUSION AND FUTURE WORK

7.1 Overview

NGS technologies are now viewed as the holy grail of exploring molecular biology. The work in this dissertation explored several aspects of NGS data including the different types of sequencing platforms, their error content, and what effects they exact on final analysis results. It further explores the error contents from various perspectives with regards to the sources of errors and the significance of knowing which error correction method to choose for a selected dataset. It depicts how the performance of NGS error correctors depends on characteristics of the NGS data. Characteristics such as the genome size of the organism, the sequence read length, the genome coverage and in a minimally explained sense, the repetitiveness of some regions of the genome.

This work is significant because it breaks down the complex nature of errors in NGS data to provide new insights and methods to alleviate erroneous base(s) issues with NGS data, especially, data from Illumina based technology, which is currently the most widely used platform. The approach employed here was to create a ground truth, in a bottom up fashion, around which all other development and analysis was built.

The study was highly experimental requiring some further studies that will build on the intuitions acquired and developed here. In this chapter, a summary of the goals and insights gained from the dissertation is presented. The contributions it makes to further enhance our knowledge of the area as well as its implications are also elaborated. Finally, we conclude with the queries tackled by this dissertation, the implications of those queries for comparative genomic analysis, and recommendations for future work.

7.2 Summary

Studying the implication of NGS errors, finding ways to mitigate these errors and verifying implication of these errors for a real-world novel organism, were the main purpose of this dissertation. Downstream analysis is complex and answering the questions that arises requires careful design, implementation, and attention to details. A systematic organization based on several prerequisite goals had to be strictly followed for the main objectives of this dissertation to be achieved. To that end, there was need to delve into a little history of how sequencing evolved. Understanding its evolvement means we can get to understand how errors became permanent fixtures of NGS data. As opposed to earlier more complicated methods, NGS methods are more error prone. This discovery prompted the need for a stringent quality control (QC) analysis to be able to achieve quality draft genome assembly. These errors constitute significant noise during downstream analysis. Having understood that NGS data contained errors in varied quantities, we investigated the methods available to correct these errors. Our focus was mainly on Illumina based data (most widely used platform) which are prone to what is known as substitution errors. In this case, a base is incorrectly replaced by another base in the sequence. Most methods available are geared towards this error types and are generally classified as k-mer or k-spectrum based methods.

Since every method implements different algorithm with varying underlying principles, further comparative analysis was carried out to investigate their performance, understand their capabilities, review their limitations, and discover ways to improve on them. Most of the existing methods are also command line based. With Linux based command line programming not been a strong forte of many molecular biologist, we explored the idea of implementing our findings as a web-based pipeline. The core idea, in this case, was to identify the most appropriate error correction algorithms for different datasets and use those to reduce the complication in execution through a web-based implementation. This provides a graphical user interface for error correction and statistical investigation allowing few parameters as possible. This pipeline known as Bioinformatics Error Correction Workflow (BECOW) was implemented in python and was the first of its kind. It provided a means for those with little to no knowledge of using these error correction tools, on Linux systems, to correct errors in their data. It also generates statistical data about the corrections made allowing the user to select the best result for their analysis.

Based on the investigation conducted, it became necessary to develop a novel approach that can deal with limitations encountered by the existing methods. Especially, limitations of dealing with reads of various length after trimming, repeat rich genomes, amount of false positive correction generated, computational resource consumption and speed. It was imperative to develop a model with the potential to alleviate the issues mentioned above.

The model, Cuckoo-filter Error Correction of Next-Generation Data (CECOND) with cuckoo-filter as its data structure was implemented. Its performance was measured comparative to existing methods using both simulated and experimental datasets. This novel algorithm produced results with about 3% false positive rate in comparison to existing methods although the overall performance in terms of f-score and precision was comparable or a little below the best performing methods. These existing methods produced results with at least 4% false positive rate. The computational resource consumption and error correction completion time were significantly reduced by about 7.85% and 6.233% respectively for CECOND. Error correction was then applied to reconstruct the genome assembly of timber rattlesnake, which is a novel organism for sequencing projects. Prior to error correction, an earlier genome assembly had been constructed with the timber rattlesnake data. This assembly was then used as a comparative approach to investigate the effect of error correction on de novo assemblies. Results indicated an improvement of the final genome assembly even in comparison to an assembly recently deposited in the NCBI data bank by the MCBIOS community. The idea is to place the genome assembly results produced in this dissertation as an updated version alongside the current existing NCBI draft genome assembly.

7.3 Contributions

Contributions of this dissertation were based on the queries or research questions that required practical answers. As mentioned earlier, it follows a systematic series of requisite research questions. The 4 main objectives (corresponding to CHAPTER III, CHAPTER IV, CHAPTER V, and CHAPTER VI) of this dissertation are highlighted here in the form of the questions while the contributions are presented as answers to those questions.

1. **Question:** Which are the most common error correction methods, how can we use them and what are their limitations if any?

   Contribution: The work presented in CHAPTER III answered these sets of questions by identifying k-mer or k-spectrum based methods as the most commonly used error correction methods. Prior to this work, no comparison of the methods had been made. We compared 6 of the methods, discovered their limitations and made recommendations to users on how best to set parameters for optimal performance of the error correctors. Through this comparative and statistical analysis, we also discovered certain inherent limitations in the false positive rates, inability to handle reads of varying lengths, ineffectiveness in correcting repeat regions of reads, high memory consumption and slow speeds of some of the error correctors. This in addition to difficulty in setting up the corrector (installation, parameter settings and ease of use) were discovered in this part of the work.

2. **Question:** Given that almost all the NGS error correctors are Linux based, can we develop a web-based pipeline that will make it easier to use for those users (especially molecular biologist), with little to no training on Linux systems?
   Contribution: Based on our analysis from CHAPTER III, we created an error correction web-based pipeline that will provide a GUI for users to input their data and supply a limited number of parameters. The pipeline runs on four (4) of the best error correctors evaluated and the novel error corrector (CECOND) we developed, to correct the user supplied data. This makes it easy for any user to correct their data without fear of using a Linux command line interface. This as well, is the first ever implementation of a web application for NGS error correction despite availability of web based pipelines for several other NGS data analysis and pipelines.
3. **Question:** With our analysis also indicating several limitations of existing NGS error correction methods, can we develop an error correction method that will alleviate some, if not all, of the associated problems we identified?

Contribution: This dissertation implemented a novel algorithm for error correction that is based on cuckoo filter data structure. Its false positive rate is based on the cuckoo filter implementation. One other distinguishing characteristics is the ability to select several thresholds across the entire data as opposed to using a single local threshold for the whole dataset. This implementation means that correction of data without uniform coverage is performed. Also reads with high repeat contents can be easily corrected as opposed to some k-mer based methods, which are incapable of correcting such datasets.

4. **Question:** Can a qualitative draft genome assembly be generated for timber rattlesnake de-novo (without any reference) if we apply error correction to it?

   Contribution: This dissertation presented a de-novo draft assembly that is of high quality as indicated by the results shown. The effect of error correction on the data was clearly visible in comparison to non-error corrected data and existing draft assembly generated from the same set of data. A hybrid approach was used in the process ensuring that we took advantage of the various data characteristics. Error correction was performed using our novel error corrector and an existing error corrector

7.4 Conclusions and Future Work

We have described some of the problems related to NGS error correction, how it affects genome-assembly and how the suit of tools we developed alleviate the associated problems. A comparative analysis of error correction methods, two-state of the art error correction tools, BECOW pipeline and CECOND error corrector and generation of de-novo timber rattlesnake (*Crotalus horridus*) were described. The work presented in this dissertation is not by any means complete and further investigation is desired. Here, we discuss some of the possible improvements to the works presented in this dissertation.

Our comparative and statistical analyses allow us to find associated problems with existing error correctors while also making recommendations to users on which methods to use. Despite this expansive view, further review is desired to include more diverse error correctors. Also, in addition to synthetic NGS dataset, running the tools on experimental dataset will give a broader overview of how these tools perform. In our opinion, this will result in a more thorough recommendation for users of such error correctors.

In addition, the development of BECOW, as the first of its kind, has eased issues associated with using the Linux command line language for error correction but requires more tweaks to ensure it is a fully developed web application. Limitations of BECOW that could be addressed ranges from increasing the size limit (currently 10 GB) of data that can be processed, incorporating novel tools as they become available, allowance for single end data to be processed (currently only paired end data is allowed), removing the compulsory requirement for a reference genome which will make statistics generation optional and modification to the interface to ensure users of the tools can be accounted for by logging into a sign on page. Addressing these issues will make BECOW more robust and effective at helping users correct their NGS data.

We further implemented CECOND error corrector which has exceptional speed and CPU memory consumption in addition to its comparable accuracy with those of existing methods. CECOND, due to the way the threshold for erroneous vs correct k-mer separation is selected, is especially tuned to correct reads with highly repetitive regions without loss of speed or increase in memory footprint. Results have shown considerable improvement over existing methods as shown in CHAPTER V. Although the results are comparable, we believe it can exceed expectations and further theoretical analysis would be necessary to improve its performance. Specifically, the algorithm implemented can be modified to correct insertion and deletion errors as newer data from other sequencing platforms like Ion Torrent becomes popular (substitution and indel errors are equally likely given such dataset). Also, allowing a user to pre-process their data completely before submission, makes it possible for us to use some contextual information provided by the associated Phred quality score. This makes the error correction process easier for us but shifts quite a bit of processing to the user. It will be worthwhile to automate such process in the future to allow CECOND pre-process data through trimming off low quality score reads. CECOND has great potential and making these modifications is recommended. To further improve the already fast run time and memory usage of the algorithm of CECOND, distributed parallelization strategy can be employed together with its already implemented multi-threading capabilities using cores

Finally, application of error correction to timber rattlesnake made a whole difference based on our result. Although the draft genome assembly generated superseded all currently available rattlesnake genomes, further work is required to get to the range of its genome size (1.9 to 2.5 Giga bases). More data needs to be sequenced using a combination of short and long NGS data platforms possibly in combination with mate-pair information. Also, the data should be properly pre-processed with errors corrected before genome assembly is started. Assembly should be performed with hybrid error correctors like MaSuRca and possible stitched with existing draft assemblies using SSPACE. This will ensure that a qualitative draft genome assembly, that is comparable to those of Burmese python and king cobra, is generated.

APPENDIX A CECOND IMPLEMENTATION GUIDE

CECOND was uniquely implemented to alleviate some issues associated with k-mer based methods. These methods are known to have issues with highly repetitive genomes because of non-uniformity of coverage across a genome, in which case, there is a drop in the correctors performance. Some of the correctors also have issues with loss of information due to how ambiguous bases are handled. The implementation of CECOND and usage is presented.

A.1 SYSTEMATIC IMPLEMENTATION GUIDE

Step 1: Take user input files and check the options. If option r =1, the file is in fasta format, proceed to next step but if r = 2, the file is in fastq format. Check the 4th line of the file. Calculate the average quality score for each line. Check that the quality score total is ≥ 20. If it is not the case, exit the process and inform the user to process their data to bring the average quality to at least 20 and resubmit the files again. CECOND assumes quality scores are in ASCII_BASE=33. It can also determine the type of sequencing data being supplied to it by the user.

Step 2: If step 1 is satisfied, take user input files and check that it contains only ACTG letters which is the second line of both fasta or fastq files. Sometimes it contains Ns. If it contains N, randomly convert the N to any of the four letters A or C or G or T.

Step3: Estimate the total number of k-mers based on value of k given by the user using the formula: Total k-mers = B ^ k. If a user gives k =8, the number of possible bases is only 4 always i.e. A, C, T and G. So, for example if k =8, then total number of k-mers = 4 ^ 8 = 65536. We can use that to set the cuckoo filter table size.

Step 4: Based on the user supplied k value, count the k-mers in an efficient method. Only k-mer counts that occur more than a given threshold of T (T is determined as shown in step 5) should be stored as fingerprints in the cuckoo filter table. Murmur hash was used to store the k-mers. Two hash tables are required and used for hashing the fingerprints and counts.

Step 5: Determine the threshold. This is done for every sequence line. So, each sequence will have a different threshold T. This is computed by first calculating the adjusted mean counts of a given sequence line e.g. If user gives k=3 and assuming sequence line = ATCGATCTCATCGACTCGCATCGTCTCATCG

COUNTS: ATC = 5, TCG = 5 CGA =2 GAT =1 TCT =2 CTC=3 TCA=2 CAT=3 GAC =1 ACT =1 CGC=1 GCA=1 CTG=1 GTC=1 TCT=1 TCA=1.

Use the count, the adjusted mean value i.e. excluding means less than minimum representation desired, calculate the harmonic mean as the threshold T for each sequence. Hence, we chose T = 2 as the threshold for the above sequence. So, any k-mer ≥ 2 is considered good or correct k-mer and those with counts < 2 are said to contain errors. These are now candidates for error correction. This process is repeated and calculated for each sequence and the threshold is used for that sequence. We rely on the closest sequences, before and after an incorrect k-mer, that satisfy this condition and see if we can turn the erroneous k-mer to the valid k-mer using the minimum number of edit operations.

Step 6: Once the count is completed and stored and the threshold is determined proceed to error correction using the k-mer counts. Take each k-mer whose count occur less than T times and compare it with the k-mer closest to it, then check the count of that k-mer in the cuckoo table. If the k-mer closest to it has a count ≥ T, then check the difference between the incorrect k-mer (invalid k-mer) and the k-mer next to it with count greater than T. Change the incorrect k-mer to the neighbor k-mer if the difference between them is one base. i.e. using hamming distance.

Step 7: After completing step 6, count the k-mers again to see if any k-mer has a count less than the threshold of T. If there is still invalid k-mers, use the same method to correct the k-mer again but this time correct it even if the difference between the k-mer and its neighbor is 2 or 3 or 4. The maxcor value of 4 (default) is the maximum number of correction that can be made. Do not allow more than m = 4 changes in a k-mer containing 10 bases i.e. a window of 10 bases (A, C, T, G characters).

Step 8: Once step 7 is completed, check through again. If there is still invalid k-mer, use exhaustive search to see if it can be corrected i.e. look for any k-mer within the sequence that is valid and correct the invalid k-mer to that k-mer.

Step 9: If after checking through all and still there is an invalid k-mer, discard the read. i.e. remove the full sequence read from the file.

Step 10: Output the result. Retrieve the result from the table, aggregate them and return the corrected sequence. The results will be the same number of files and input format given by the user. i.e. if two fastq files are given, the output will be two fastq files with the corrections performed. If the input is 2 fasta files, then the output will be two fasta files with corrections performed. Fastq file can end with fastq or fq while fasta files end with fasta or fa.

APPENDIX B EXPLORATORY TABLES AND FIGURES

B.1.1 Representation of TP, FP and FN

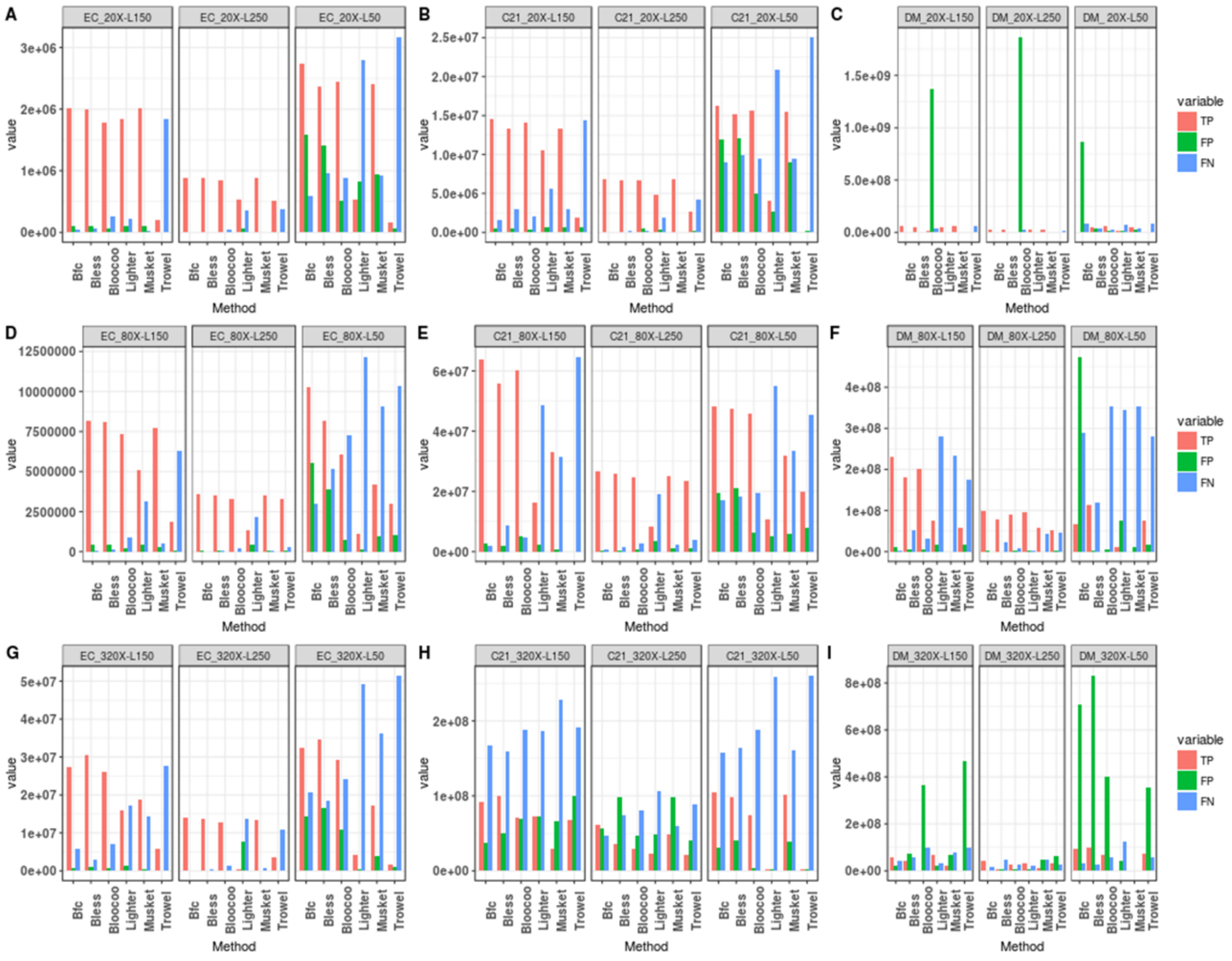


Figure B.1 Performance metrics of six k-mer spectrum-based error correctors on 27 synthetic datasets. Eco = *E. coli*; Chr21 = Human chromosome 21; Dme = *D. melanogaster;* 20X, 80X and 320X = 20-, 80- and 320-fold coverage; L50, L150 and L300 = read length of 50, 150 and 300 bp. Metrics measurements for TP, FP and FN.

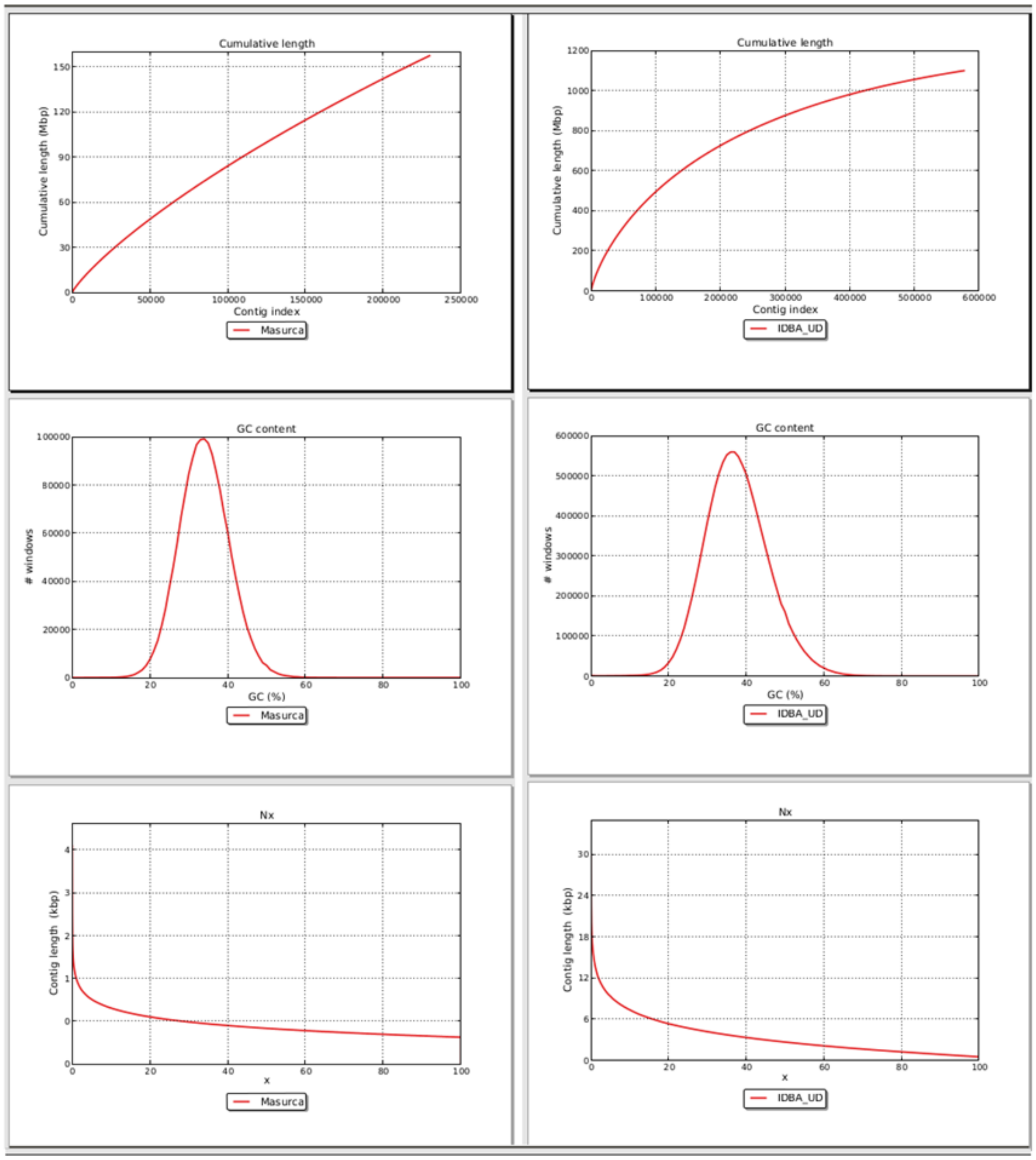

Cumulative length
Cumulative length (Mbp)
Contig index
Masurca
IDBA_UD
GC content
# windows
GC (%)
Nx
Contig length (kbp)
x

Figure B.2 QUAST comparison of MaSuRCA vs IDBA-UD generated assembly. MaSuRCA showed better performance over IDBA-UD and was used for genome assembly. It is also a hybrid assembler that can handle data from multiple platforms.

BIBLIOGRAPHY

[1] J. D. Watson and F. H. C. Crick, “Molecular structure of nucleic aids: A structure for deoxyribose nucleic acid,” *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.

[2] V. Precone *et al.*, “Cracking the Code of Human Diseases Using Next-Generation Sequencing : Applications , Challenges , and Perspectives,” *Biomed Res. Int.*, vol. 2015, 2015.

[3] J. Henson, G. Tischler, and Z. Ning, “Next-generation sequencing and large genome assemblies,” *Pharmacogenomics*, vol. 13, no. 8, pp. 901–915, 2012.

[4] H. P. J. Buermans and J. T. den Dunnen, “Next generation sequencing technology: Advances and applications,” *Biochim. Biophys. Acta*, vol. 1842, no. 10, pp. 1932–1941, 2014.

[5] L. Liu *et al.*, “Comparison of next-generation sequencing systems,” *Journal of Biomedicine and Biotechnology*, vol. 2012. 2012.

[6] J. Shendure and H. Ji, “Next-generation DNA sequencing,” *Nat Biotechnol*, vol. 26, pp. 1135–1145, 2008.

[7] L. T. C. Franc: A, E. Carrilho, and T. B. L. Kist, “A review of DNA sequencing techniques,” *Q. Rev. Biophys.*, vol. 35, no. 2, pp. 169–200, 2002.

[8] E. E. Schadt, S. Turner, and A. Kasarskis, “A window into third-generation sequencing,” *Hum. Mol. Genet.*, vol. 19, no. R2, 2010.

[9] T. P. Niedringhaus, D. Milanova, M. B. Kerby, M. P. Snyder, and A. E. Barron, “Landscape of next-generation sequencing technologies,” *Analytical Chemistry*, vol. 83, no. 12. pp. 4327–4341, 2011.

[10] C. S. Pareek, R. Smoczynski, and A. Tretyn, “Sequencing technologies and

genome sequencing," *Journal of Applied Genetics*, vol. 52, no. 4. pp. 413–435, 2011.

[11] I. G. Gut, "New sequencing technologies," *Clin. Transl. Oncol.*, vol. 15, no. 11, pp. 879–881, 2013.

[12] R. L. Sinsheimer, "Purification and properties of bacteriophage fX174," *J. Mol. Biol.*, vol. 1, pp. 37–42, 1959.

[13] W. R. Holley *et al.*, "Struture of a ribonuclei acid," *Science*, vol. 147, no. 3664, pp. 1462–1465, 1965.

[14] R. Wu and A. D. Kaiser, "Mapping the 5'-terminal nucleotides of the DNA of bacteriophage lambda and related phages.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 57, no. 1, pp. 170–7, Jan. 1967.

[15] R. Wu and A. D. Kaiser, "Structure and base sequence in the cohesive ends of bacteriophage lambda DNA," *J. Mol. Biol.*, vol. 35, no. 3, pp. 523–537, 1968.

[16] A. D. Kaiser and R. Wu, "Structure and function of DNA cohesive ends.," *Cold Spring Harb. Symp. Quant. Biol.*, vol. 33, pp. 729–734, 1968.

[17] R. Wu and E. Taylor, "Nucleotide sequence analysis of DNA. II. Complete nucleotide sequence of the cohesive ends of bacteriophage lambda DNA.," *J. Mol. Biol.*, vol. 57, no. 3, pp. 491–511, 1971.

[18] R. Padmanabhan, R. Padmanabhan, and R. Wu, "Nucleotide sequence analysis of DNA. IX. Use of oligonucleotides of defined sequence as primers in DNA sequence analysis," *Biochem. Biophys. Res. Commun.*, vol. 48, no. 5, pp. 1295–1302, 1972.

[19] H. O. Smith, "NUCLEOTIDE SEQUENCE SPECIFICITY OF RESTRICTION

ENDONUCLEASES," 1978.

[20] H. O. Smith and K. W. Welcox, "A Restriction enzyme from Hemophilus influenzae I," *J. Mol. Biol.*, vol. 51, no. 2, pp. 379–391, 1970.

[21] T. J. Kelly and H. O. Smith, "A restriction enzyme from Hemophilus influenzae II," *J. Mol. Biol.*, vol. 51, no. 2, pp. 393–409, 1970.

[22] J. H. Middleton, M. H. Edgell, and C. A. Hutchison, "Specific fragments of phi X174 deoxyribonucleic acid produced by a restriction enzyme from Haemophilus aegyptius, endonuclease Z.," *J. Virol.*, vol. 10, no. 1, pp. 42–50, 1972.

[23] M. Kaus-Drobek *et al.*, "Restriction endonuclease MvaI is a monomer that recognizes its target sequence asymmetrically," *Nucleic Acids Res.*, vol. 35, no. 6, pp. 2035–2046, 2007.

[24] F. Sanger and A. R. Coulson, "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase," *J. Mol. Biol.*, vol. 94, no. 3, 1975.

[25] F. Sanger, "The Croonian Lecture, 1975: Nucleotide Sequences in DNA," *Proc. R. Soc. B Biol. Sci.*, vol. 191, no. 1104, pp. 317–333, 1975.

[26] F. Sanger, "DETERMINATION OF NUCLEOTIDE SEQUENCES IN DNA," 1980.

[27] a M. Maxam and W. Gilbert, "A new method for sequencing DNA.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 74, no. 2, pp. 560–4, 1977.

[28] F. Sanger, S. Nicklen, and a R. Coulson, "DNA sequencing with chain-terminating inhibitors.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 74, no. 12, pp. 5463–7, 1977.

[29] C. A. Hutchison, "DNA sequencing: Bench to bedside and beyond," *Nucleic Acids*

Res., vol. 35, no. 18, pp. 6227–6237, 2007.

[30] K. Mullis, F. Faloona, S. Scharf, R. Saiki, G. Horn, and H. Erlich, *Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction.*, vol. 51 Pt 1. 1986.

[31] L. Smith *et al.*, “Fluorescence detection in automated DNA sequence analysis.,” *Nature*, vol. 321, no. 6071, pp. 674–679, 1986.

[32] M. Ronaghi, S. Karamohamed, B. Pettersson, M. Uhlén, and P. L. Nyrén, “Real-Time DNA Sequencing Using Detection of Pyrophosphate Release,” *Anal. Biochem.*, vol. 242, pp. 84–89, 1996.

[33] M. Ronaghi, M. Uhlén, and P. Nyrén, “PyroSequencing: A {DNA} sequencing method based on real-time pyrophosphate detection,” *Science (80-.).*, vol. 281, pp. 363–365, 1998.

[34] S. Brenner *et al.*, “Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays.,” *Nat. Biotechnol.*, vol. 18, no. 6, pp. 630–634, 2000.

[35] R. D. Mitra, J. Shendure, J. Olejnik, E. Krzymanska-Olejnik, and G. M. Church, “Fluorescent in situ sequencing on polymerase colonies,” *Anal. Biochem.*, vol. 320, no. 1, pp. 55–65, 2003.

[36] “Sequencing by Ligation.” [Online]. Available: https://binf.snipcademy.com/lessons/ngs-techniques/sequencing-by-ligation. [Accessed: 03-Jun-2017].

[37] “Miniaturized, high-throughput nucleic acid analysis,” 2010.

[38] J. C. Wooley, A. Godzik, and I. Friedberg, “A primer on metagenomics,” *PLoS*

Computational Biology, vol. 6, no. 2. 2010.

[39] T. S. Seo, X. Bai, H. Ruparel, Z. Li, N. J. Turro, and J. Ju, “Photocleavable fluorescent nucleotides for DNA sequencing on a chip constructed by site-specific coupling chemistry.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 101, no. 15, pp. 5488–93, 2004.

[40] J. Guo, L. Yu, N. J. Turro, and J. Ju, “An integrated system for DNA sequencing by synthesis using novel nucleotide analogues.,” *Acc. Chem. Res.*, vol. 43, no. 4, pp. 551–63, Apr. 2010.

[41] “Human Genome Project: Illumina Sequencing.” [Online]. Available: http://itghumangenomeprojectwallpapars.blogspot.com/2012/12/illumina-sequencing.html. [Accessed: 28-Jun-2017].

[42] J. Shendure *et al.*, “Accurate multiplex polony sequencing of an evolved bacterial genome.,” *Science*, vol. 309, no. 5741, pp. 1728–32, 2005.

[43] I. Braslavsky, B. Hebert, E. Kartalov, and S. R. Quake, “Sequence information can be obtained from single DNA molecules.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 100, no. 7, pp. 3960–4, 2003.

[44] T. D. Harris *et al.*, “Single-molecule DNA sequencing of a viral genome.,” *Science*, vol. 320, no. 5872, pp. 106–9, 2008.

[45] J. Bowers *et al.*, “Virtual terminator nucleotides for next-generation DNA sequencing.,” *Nat. Methods*, vol. 6, no. 8, pp. 593–5, 2009.

[46] B. A. Flusberg *et al.*, “Direct detection of DNA methylation during single-molecule, real-time sequencing.,” *Nat. Methods*, vol. 7, no. 6, pp. 461–5, 2010.

[47] J. Li, D. Stein, C. McMullan, D. Branton, M. J. Aziz, and J. A. Golovchenko,

"Ion-beam sculpting at nanometre length scales," *Nature*, vol. 412, no. 6843, pp. 166–169, Jul. 2001.

[48] C. Dekker, "Solid-state nanopores," *Nat. Nanotechnol.*, vol. 2, no. 4, pp. 209–215, 2007.

[49] N. J. Loman and A. R. Quinlan, "Poretools: A toolkit for analyzing nanopore sequence data," *Bioinformatics*, vol. 30, no. 23, pp. 3399–3401, 2014.

[50] D. Sims, I. Sudbery, N. E. Ilott, A. Heger, and C. P. Ponting, "Sequencing depth and coverage: key considerations in genomic analyses," *Nat. Rev. Genet.*, vol. 15, no. 2, pp. 121–132, Jan. 2014.

[51] P. Richterich, "Estimation of errors in &quot;raw&quot; DNA sequences: a validation study.," *Genome Res.*, vol. 8, no. 3, pp. 251–9, Mar. 1998.

[52] a Lario, a González, and G. Dorado, "Automated laser-induced fluorescence DNA sequencing: equalizing signal-to-noise ratios significantly enhances overall performance.," *Anal. Biochem.*, vol. 247, no. 1, pp. 30–3, 1997.

[53] B. B. Rosenblum *et al.*, "New dye-labeled terminators for improved DNA sequencing patterns.," *Nucleic Acids Res.*, vol. 25, no. 22, pp. 4500–4504, 1997.

[54] K. Robasky, N. E. Lewis, and G. M. Church, "The role of replicates for error mitigation in next-generation sequencing," *Nat. Rev. Genet.*, vol. 15, no. 1, pp. 56–62, Dec. 2013.

[55] S. Hoffmann *et al.*, "Fast mapping of short sequences with mismatches, insertions and deletions using index structures," *PLoS Comput. Biol.*, vol. 5, no. 9, 2009.

[56] S. M. Huse and D. B. M. Welch, "Accuracy and Quality of Massively Parallel DNA Pyrosequencing," in *Handbook of Molecular Microbial Ecology I:*

Metagenomics and Complementary Approaches, 2011, pp. 149–155.

[57] J. Archer, G. Baillie, S. J. Watson, P. Kellam, A. Rambaut, and D. L. Robertson, "Analysis of high-depth sequence data for studying viral diversity: a comparison of next generation sequencing platforms using Segminator II."

[58] M. Kircher, U. Stenzel, and J. Kelso, "Improved base calling for the Illumina Genome Analyzer using machine learning strategies.," *Genome Biol.*, vol. 10, no. 8, p. R83, 2009.

[59] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer, "Substantial biases in ultra-short read data sets from high-throughput DNA sequencing," *Nucleic Acids Res.*, vol. 36, no. 16, 2008.

[60] M. L. Metzker, "Sequencing technologies - the next generation.," *Nat. Rev. Genet.*, vol. 11, no. 1, pp. 31–46, 2010.

[61] A. J. Berno, "A graph theoretic approach to the analysis of DNA sequencing data.," *Genome Res.*, vol. 6, no. 2, pp. 80–91, Feb. 1996.

[62] A. Lario, A. González, and G. Dorado, "Automated Laser-Induced Fluorescence DNA Sequencing: Equalizing Signal-to-Noise Ratios Significantly Enhances Overall Performance," *Anal. Biochem.*, vol. 247, no. 1, pp. 30–33, Apr. 1997.

[63] J. Z. Sanders *et al.*, "Imaging as a tool for improving length and accuracy of sequence analysis in automated fluorescence-based DNA sequencing," *Electrophoresis*, vol. 12, no. 1, pp. 3–11, Jan. 1991.

[64] K. D. Hansen, S. E. Brenner, and S. Dudoit, "Biases in Illumina transcriptome sequencing caused by random hexamer priming," *Nucleic Acids Res.*, vol. 38, no. 12, pp. e131–e131, Jul. 2010.

[65] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig, “A Parallel Algorithm for Error Correction in High-Throughput Short-Read Data on CUDA-Enabled Graphics Hardware,” *J. Comput. Biol.*, vol. 17, no. 4, pp. 603–615, Apr. 2010.

[66] P. A. Pevzner, H. Tang, and M. S. Waterman, “An Eulerian path approach to DNA fragment assembly.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 98, no. 17, pp. 9748–53, Aug. 2001.

[67] M. Shugay *et al.*, “Towards error-free profiling of immune repertoires,” *Nat. Methods*, vol. 11, no. 6, pp. 653–655, May 2014.

[68] D. I. Lou *et al.*, “High-throughput DNA sequencing errors are reduced by orders of magnitude using circle sequencing,” *Proc. Natl. Acad. Sci.*, vol. 110, no. 49, pp. 19872–19877, Dec. 2013.

[69] T. Buschmann and L. V Bystrykh, “Levenshtein error-correcting barcodes for multiplexed DNA sequencing,” *BMC Bioinformatics*, vol. 14, no. 1, p. 272, 2013.

[70] M. J. Chaisson and P. A. Pevzner, “Short read fragment assembly of bacterial genomes,” *Genome Res.*, vol. 18, no. 2, pp. 324–330, Feb. 2008.

[71] J. Schroder, H. Schroder, S. J. Puglisi, R. Sinha, and B. Schmidt, “SHREC: a short-read error correction method,” *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, Sep. 2009.

[72] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, “Quake: quality-aware detection and correction of sequencing errors,” *Genome Biol.*, vol. 11, no. 11, p. R116, 2010.

[73] L. Ilie, F. Fazayeli, and S. Ilie, “HiTEC: accurate error correction in high-throughput sequencing data,” *Bioinformatics*, vol. 27, no. 3, pp. 295–302, Feb. 2011.

[74] M. Tahir, M. Sardaraz, A. A. Ikram, and H. Bajwa, “Review of Genome Sequence Short Read Error Correction Algorithms,” *Am. J. Bioinforma. Res.*, vol. 3, no. 1, pp. 1–9, 2013.

[75] X. Yang, S. P. Chockalingam, and S. Aluru, “A survey of error-correction methods for next-generation sequencing,” *Brief. Bioinform.*, vol. 14, no. 1, pp. 56–66, Jan. 2013.

[76] P. A. Pevzner, H. Tang, and M. S. Waterman, “A new approach to fragment assembly in DNA sequencing,” in *Proceedings of the fifth annual international conference on Computational biology - RECOMB ’01*, 2001, pp. 256–267.

[77] M. Chaisson, P. Pevzner, and H. Tang, “Fragment assembly with short reads,” *Bioinformatics*, vol. 20, no. 13, pp. 2067–2074, Sep. 2004.

[78] W. Qu, S.-I. Hashimoto, and S. Morishita, “Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing.,” *Genome Res.*, vol. 19, no. 7, pp. 1309–15, Jul. 2009.

[79] E. Wijaya, M. C. Frith, Y. Suzuki, and P. Horton, “Recount: expectation maximization based error correction tool for next generation sequencing data.,” *Genome Inform.*, vol. 23, no. 1, pp. 189–201, Oct. 2009.

[80] X. Yang, K. S. Dorman, and S. Aluru, “Reptile: representative tiling for short read error correction,” *Bioinformatics*, vol. 26, no. 20, pp. 2526–2533, Oct. 2010.

[81] H.-S. Le, M. H. Schulz, B. M. McCauley, V. F. Hinman, and Z. Bar-Joseph, “Probabilistic error correction for RNA sequencing,” *Nucleic Acids Res.*, vol. 41, no. 10, pp. e109–e109, May 2013.

[82] J. Butler *et al.*, “ALLPATHS: De novo assembly of whole-genome shotgun

microreads," *Genome Res.*, vol. 18, no. 5, pp. 810–820, Feb. 2008.

[83] R. Li *et al.*, "De novo assembly of human genomes with massively parallel short read sequencing.," *Genome Res.*, vol. 20, no. 2, pp. 265–72, Feb. 2010.

[84] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures.," *Genome Res.*, vol. 22, no. 3, pp. 549–56, Mar. 2012.

[85] A. Bankevich *et al.*, "SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing.," *J. Comput. Biol.*, vol. 19, no. 5, pp. 455–77, May 2012.

[86] S. I. Nikolenko, A. I. Korobeynikov, and M. A. Alekseyev, "BayesHammer: Bayesian clustering for error correction in single-cell sequencing," *BMC Genomics*, vol. 14, no. Suppl 1, p. S7, 2013.

[87] "Kmer Spectrum Primer."

[88] L. Salmela, "Correction of sequencing errors in a mixed set of reads," *Bioinformatics*, vol. 26, no. 10, pp. 1284–1290, May 2010.

[89] L. Ilie and M. Molnar, "RACER: Rapid and accurate correction of errors in reads," *Bioinformatics*, vol. 29, no. 19, pp. 2490–2493, Oct. 2013.

[90] M. T. Tammi, E. Arner, E. Kindlund, and B. Andersson, "Correcting errors in shotgun sequences.," *Nucleic Acids Res.*, vol. 31, no. 15, pp. 4663–72, Aug. 2003.

[91] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins.," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–53, Mar. 1970.

[92] D. B. Jaffe *et al.*, "Whole-Genome Sequence Assembly for Mammalian Genomes: Arachne 2," *Genome Res.*, vol. 13, no. 1, pp. 91–96, Jan. 2003.

[93] L. Salmela and J. Schr??der, “Correcting errors in short reads by multiple alignments,” *Bioinformatics*, vol. 27, no. 11, pp. 1455–1461, 2011.

[94] W. C. Kao, A. H. Chan, and Y. S. Song, “ECHO: A reference-free short-read error correction algorithm,” *Genome Res.*, vol. 21, no. 7, pp. 1181–1192, 2011.

[95] C.-S. Chin *et al.*, “Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data,” *Nat. Methods*, vol. 10, no. 6, pp. 563–569, May 2013.

[96] A. Alic, A. Tomás, J. Salavert, I. Medina, and I. Blanquer, “Robust Error Correction for De Novo Assembly via Spectral Partitioning and Sequence Alignment.”

[97] X. Yin, Z. Song, K. Dorman, and A. Ramamoorthy, “PREMIER - PRobabilistic Error-correction using Markov Inference in Errored Reads,” Feb. 2013.

[98] L. Salmela and E. Rivals, “LoRDEC: accurate and efficient long read error correction,” *Bioinformatics*, vol. 30, no. 24, pp. 3506–3514, Dec. 2014.

[99] T. Hackl, R. Hedrich, J. Schultz, and F. Forster, “proovread: large-scale high-accuracy PacBio correction through iterative short read consensus,” *Bioinformatics*, vol. 30, no. 21, pp. 3004–3011, Nov. 2014.

[100] K. F. Au, J. G. Underwood, L. Lee, W. H. Wong, and B. Walenz, “Improving PacBio Long Read Accuracy by Short Read Alignment,” *PLoS One*, vol. 7, no. 10, p. e46679, Oct. 2012.

[101] S. Koren *et al.*, “Hybrid error correction and de novo assembly of single-molecule sequencing reads,” *Nat. Biotechnol.*, vol. 30, no. 7, pp. 693–700, Jul. 2012.

[102] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, “Replacing suffix trees with

enhanced suffix arrays," *J. Discret. Algorithms*, vol. 2, pp. 53–86, 2004.

[103] S. M. Kielbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith, "Adaptive seeds tame genomic sequence comparison," *Genome Res.*, vol. 21, no. 3, pp. 487–493, Mar. 2011.

[104] M. Vyverman, J. De Schrijver, W. Van Criekinge, P. Dawyndt, and V. Fack, "ACCURATE LONG READ MAPPING USING ENHANCED SUFFIX ARRAYS."

[105] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt, "essaMEM: finding maximal exact matches using enhanced sparse suffix arrays," *Bioinformatics*, vol. 29, no. 6, pp. 802–804, Mar. 2013.

[106] Y. Ye, J.-H. Choi, and H. Tang, "RAPSearch: a fast protein similarity search tool for short reads."

[107] G. Gonnella and S. Kurtz, "Readjoiner: a fast and memory efficient string graph-based sequence assembler."

[108] D. Hernandez, P. Francois, L. Farinelli, M. Osteras, and J. Schrenzel, "De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer," *Genome Res.*, vol. 18, no. 5, pp. 802–809, Feb. 2008.

[109] B. Genomics, S. Kurtz, A. Narechania, J. C. Stein, and D. Ware, "A new method to compute K-mer frequencies and its application to annotate large repetitive plant genomes," *BMC Genomics*, vol. 9, no. 9, 2008.

[110] S. Hazelhurst and Z. Lipták, "KABOOM! A new suffix array based algorithm for clustering expression data," vol. 27, no. 24, pp. 3348–335510, 2011.

[111] R. Homann, D. Fleer, R. Giegerich, and M. Rehmsmeier, "mkESA: enhanced

suffix array construction tool," *Bioinformatics*, vol. 25, no. 8, pp. 1084–1085, Apr. 2009.

[112] T. H. Cormen, *Introduction to algorithms*. MIT Press, 2009.

[113] "Suffix tree and suffix array for string matching." [Online]. Available: http://homes.soic.indiana.edu/yye/lab/teaching/spring2014-C343/suffix.php. [Accessed: 03-Jun-2017].

[114] M. Nicolaidis, "Design for soft error mitigation," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 405–418, Sep. 2005.

[115] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors."

[116] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509.

[117] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "LNCS 4168 - An Improved Construction for Counting Bloom Filters," 2006.

[118] M. Mitzenmacher, "Compressed Bloom Filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, 2002.

[119] Y. Heo, X.-L. Wu, D. Chen, J. Ma, and W.-M. Hwu, "BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads," vol. 30, no. 10, pp. 1354–1362, 2014.

[120] Y. Liu, J. Schroder, and B. Schmidt, "Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data," *Bioinformatics*, vol. 29, no. 3, pp. 308–315, Feb. 2013.

[121] E. C. Lim, J. Müller, J. Hagmann, S. R. Henz, S. T. Kim, and D. Weigel, "Trowel: A fast and accurate error correction module for Illumina sequencing reads,"

Bioinformatics, vol. 30, no. 22, pp. 3264–3265, 2014.

[122] E. Drezen *et al.*, "GATB: Genome Assembly &amp; Analysis Tool Box.," *Bioinformatics*, vol. 30, no. 20, pp. 2959–61, Oct. 2014.

[123] H. Li, "BFC: correcting Illumina sequencing errors," *Bioinformatics*, vol. 31, no. 17, pp. 2885–2887, Sep. 2015.

[124] L. Song, L. Florea, and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome Biol.*, vol. 15, no. 11, p. 509, Nov. 2014.

[125] P. Lin, H. Deng, F. Wang, and W. Tan, "A study on d-left counting bloom filter for dynamic packets filtering," *Information*, vol. 14, no. 4, pp. 1353–1361, 2011.

[126] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo Filter," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies - CoNEXT '14*, 2014, vol. 38, pp. 75–88.

[127] R. Pagh and F. Friche Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, pp. 122–144, 2004.

[128] G. Rizk, D. Lavenier, and R. Chikhi, "DSK: K-mer counting with very low memory usage," *Bioinformatics*, vol. 29, no. 5, pp. 652–653, 2013.

[129] S. Deorowicz, M. Kokot, S. Grabowski, and A. Debudaj-Grabysz, "KMC 2: Fast and resource-frugal k-mer counting," *Bioinformatics*, vol. 31, no. 10, pp. 1569–1576, 2014.

[130] Y. Liu, B. Schmidt, and D. L. Maskell, "Cushaw: A cuda compatible short read aligner to large genomes based on the burrows-wheeler transform,"

Bioinformatics, vol. 28, no. 14, pp. 1830–1837, 2012.

[131] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An Improved Construction for Counting Bloom Filters," *Lect. Notes Comput. Sci.*, vol. 4168, p. 684, 2006.

[132] R. Ekblom and J. B. W. Wolf, "A field guide to whole-genome sequencing, assembly and annotation," *Evol. Appl.*, vol. 7, no. 9, pp. 1026–1042, Nov. 2014.

[133] Z. Zhao, J. Yin, Y. Li, W. Xiong, and Y. Zhan, "An efficient hybrid approach to correcting errors in short reads," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* 2011, vol. 6820 LNAI, pp. 198–210.

[134] D. Mapleson, G. Garcia Accinelli, G. Kettleborough, J. Wright, and B. J. Clavijo, "KAT: A K-mer Analysis Toolkit to quality control NGS datasets and genome assemblies.," *Bioinformatics*, p. btw663, 2016.

[135] M. J. Chaisson, D. Brinza, and P. A. Pevzner, "De novo fragment assembly with short mate-paired reads: Does the read length matter?," *Genome Res.*, vol. 19, no. 2, pp. 336–346, 2009.

[136] X. Zhao, L. E. Palmer, R. Bolanos, C. Mircean, D. Fasulo, and G. M. Wittenberg, "EDAR: an efficient error detection and removal algorithm for next generation sequencing data.," *J. Comput. Biol.*, vol. 17, no. 11, pp. 1549–60, 2010.

[137] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[138] R. Chikhi and P. Medvedev, "Informed and automated k-mer size selection for genome assembly," *Bioinformatics*, vol. 30, no. 1, pp. 31–37, 2014.

[139] W. Huang, L. Li, J. R. Myers, and G. T. Marth, “ART: A next-generation sequencing read simulator,” *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[140] I. Akogwu *et al.*, “A comparative study of k-spectrum-based error correction methods for next-generation sequencing data analysis,” *Hum. Genomics*, vol. 10, no. S2, p. 20, 2016.

[141] H. Li, “BFC: Correcting Illumina sequencing errors,” *Bioinformatics*, vol. 31, no. 17, pp. 2885–2887, 2015.

[142] Y. Heo, A. Ramachandran, W. M. Hwu, J. Ma, and D. Chen, “BLESS 2: Accurate, memory-efficient and fast error correction method,” *Bioinformatics*, vol. 32, no. 15, pp. 2369–2371, 2016.

[143] R. C. Littell, G. A. Milliken, W. W. Stroup, and R. D. Wolfinger, *SAS system for mixed models*. 1996.

[144] D. Montgomery, “Design and Analysis of Experiments,” *Jhon Wiley & Sons, Inc*, vol. 5th. p. 684, 2001.

[145] M. Molnar and L. Ilie, “Correcting Illumina data,” *Brief. Bioinform.*, vol. 16, no. 4, pp. 588–599, 2014.

[146] D. Laehnemann, A. Borkhardt, and A. C. McHardy, “Denoising DNA deep sequencing data-high-throughput sequencing errors and their correction,” *Brief. Bioinform.*, vol. 17, no. 1, pp. 154–179, 2016.

[147] K. Sameith, J. G. Roscito, and M. Hiller, “Iterative error correction of long sequencing reads maximizes accuracy and improves contig assembly,” *Brief. Bioinform.*, no. October 2015, p. bbw003, 2016.

[148] J. T. Simpson and R. Durbin, “Efficient de novo assembly of large genomes using

compressed data structures," *Genome Res.*, vol. 22, no. 3, pp. 549–556, 2012.

[149] C. Quince *et al.*, "Accurate determination of microbial diversity from 454 pyrosequencing data.," *Nat. Methods*, vol. 6, no. 9, pp. 639–41, 2009.

[150] A. S. Alic, D. Ruzafa, J. Dopazo, and I. Blanquer, "Objective review of de novo stand-alone error correction methods for NGS data," *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, vol. 6, no. 2, pp. 111–146, 2016.

[151] R. Leinonen, H. Sugawara, M. Shumway, and International Nucleotide Sequence Database Collaboration, "The sequence read archive.," *Nucleic Acids Res.*, vol. 39, no. Database issue, pp. D19-21, Jan. 2011.

[152] A. V. Zimin, G. Marcais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke, "The MaSuRCA genome assembler," *Bioinformatics*, vol. 29, no. 21, pp. 2669–2677, Nov. 2013.

[153] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community."

[154] E. C. Berglund, A. Kiialainen, and A.-C. Syv?nen, "Next-generation sequencing technologies and applications for human genetic history and forensics," *Investig. Genet.*, vol. 2, no. 1, p. 23, 2011.

[155] A. Pagh, R. Pagh, and S. S. Rao, "An Optimal Bloom Filter Replacement *."

[156] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo Filter," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies - CoNEXT '14*, 2014, vol. 38, pp. 75–88.

[157] P. E. C. Compeau, P. A. Pevzner, and G. Tesler, "How to apply de Bruijn graphs

to genome assembly," *Nat. Biotechnol.*, vol. 29, no. 11, pp. 987–991, 2011.

[158] S. S. Sindi, B. R. Hunt, and J. A. Yorke, "Duplication count distributions in DNA sequences," *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, vol. 78, no. 6, 2008.

[159] R. C. Edgar, "MUSCLE: Multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Res.*, vol. 32, no. 5, pp. 1792–1797, 2004.

[160] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no. 6, pp. 764–770, Mar. 2011.

[161] M. R. Crusoe *et al.*, "The khmer software package: enabling efficient nucleotide sequence analysis," *F1000Research*, 2015.

[162] P. Audano and F. Vannberg, "KAnalyze: A fast versatile pipelined K-mer toolkit," *Bioinformatics*, vol. 30, no. 14, pp. 2070–2072, 2014.

[163] T. C. Conway and A. J. Bromage, "Succinct data structures for assembling large genomes," *Bioinformatics*, vol. 27, no. 4, pp. 479–486, 2011.

[164] Y. Li and X. Yan, "MSPKmerCounter: A Fast and Memory Efficient Approach for K-mer Counting," vol. 0, no. 0, pp. 1–7, 2013.

[165] P. Greenfield, K. Duesing, A. Papanicolaou, and D. C. Bauer, "Blue: Correcting sequencing errors using consensus and context," *Bioinformatics*, vol. 30, no. 19, pp. 2723–2732, 2014.

[166] K. Nakamura *et al.*, "Sequence-specific error profile of Illumina sequencers," *Nucleic Acids Res.*, vol. 39, no. 13, 2011.

[167] X. Yang, S. Aluru, and K. S. Dorman, "Repeat-aware modeling and correction of short read errors.," *BMC Bioinformatics*, vol. 12 Suppl 1, no. Suppl 1, p. S52, Feb.

2011.

[168] R. W. Hamming, “Error Detecting and Error Correcting Codes,” *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950.

[169] I. Akogwu, N. Wang, C. Zhang, Hwanseok Choi, H. Hong, and P. Gong, “Factorial analysis of error correction performance using simulated next-generation sequencing data,” in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2016, pp. 1164–1169.

[170] S. L. Salzberg *et al.*, “GAGE: A critical evaluation of genome assemblies and assembly algorithms,” *Genome Res.*, vol. 22, no. 3, pp. 557–567, 2012.

[171] H. Li, “Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM,” Mar. 2013.

[172] G. Marçais and C. Kingsford, “A fast, lock-free approach for efficient parallel counting of occurrences of k-mers,” *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011.

[173] R. Luo *et al.*, “Erratum: SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler,” *Gigascience*, vol. 4, no. 1, p. 30, 2015.

[174] Genome 10K Community of Scientists, “Genome 10K: A Proposal to Obtain Whole-Genome Sequence for 10 000 Vertebrate Species,” *J. Hered.*, vol. 100, no. 6, pp. 659–674, Nov. 2009.

[175] K. R. Bradnam *et al.*, “Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species,” Jan. 2013.

[176] A. M. Bolger, M. Lohse, and B. Usadel, “Trimmomatic: a flexible trimmer for Illumina sequence data.,” *Bioinformatics*, vol. 30, no. 15, pp. 2114–20, Aug. 2014.

[177] R. K. Patel, M. Jain, C. Schlotterer, P. Aboyoun, and H. Pages, “NGS QC Toolkit: A Toolkit for Quality Control of Next Generation Sequencing Data,” *PLoS One*, vol. 7, no. 2, p. e30619, Feb. 2012.

[178] H. Lab, “FASTX Toolkit,” *http://hannonlab.cshl.edu/fastx_toolkit/index.html*.

[179] Andrews, “FastQC A Quality Control tool for High Throughput Sequence Data,” *http://www.bioinformatics.babraham.ac.uk/projects/fastqc/*.

[180] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, “QUAST: Quality assessment tool for genome assemblies,” *Bioinformatics*, vol. 29, no. 8, pp. 1072–1075, 2013.

[181] Y. Peng, H. C. M. Leung, S. M. Yiu, and F. Y. L. Chin, “IDBA-UD: A de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth,” *Bioinformatics*, vol. 28, no. 11, pp. 1420–1428, 2012.

[182] C. TA and et al, “Sequencing the genome of the Burmese python\r(Python molurus bivittatus) as a model for studying\rextreme adaptations in snakes,” *Genome Biol*, vol. 12, p. 406, 2011.

[183] F. J. Vonk *et al.*, “The king cobra genome reveals dynamic gene evolution and adaptation in the snake venom system.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 110, no. 51, pp. 20651–20656, 2013.

[184] M. C. Schatz, A. L. Delcher, and S. L. Salzberg, “Assembly of large genomes using second-generation sequencing,” *Genome Research*, vol. 20, no. 9. pp. 1165–1173, 2010.

[185] M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler, and W. Pirovano, “Scaffolding pre-assembled contigs using SSPACE,” *Bioinformatics*, vol. 27, no. 4, pp. 578–579, Feb. 2011.