Spring 2012

Utilization of Probabilistic Models in Short Read Assembly from Second-Generation Sequencing

Matthew W. Segar
Bucknell University, mws022@bucknell.edu

Utilization of Probabilistic Models in Short Read Assembly from Second-Generation Sequencing

by

Matthew W. Segar

A Thesis

Presented to the Faculty of
Bucknell University
In partial Fulfillment of the Requirements for the Degree of
Bachelor of Arts with Honors in Computer Science
May 10, 2012

Approved: _____

Dr. Brian King
Thesis Advisor

_____

Dr. Daniel Hyde
Acting Chair, Department of Computer Science

Acknowledgements

This thesis would not have been possible without the support of many people. I would like to take this time to thank these individuals for their continued encouragement and belief in me throughout my time at Bucknell:

- First and foremost, Dr. Brian King, for all his guidance, help, and patience in introducing me to the crazy field of bioinformatics. I've enjoyed the time we've spent rambling on about our lives and I appreciate all you knowledge you've instilled upon me. I can't thank you enough.

- Peg Cronin, for her knowledge and expertise in helping me write this thesis. I came to you as a nervous and timid writer and you taught me techniques that I will carry with me for years to come. This thesis would not have been possible without your help.

- Professor L. Felipe Perrone and Professor Sharon Garthwaite, for helping me understand the thesis defense process and taking the time out of your busy schedules to be my faculty readers.

- Aurimas Liutikas, for helping me immensely the last 4 years. I've thoroughly enjoyed talking technology with you and I can't thank you enough for all the support you've given me.

- Family and friends, for supporting me throughout this entire thesis process.

Table of Contents

List of Tables

List of Figures

Abstract

With the advent of cheaper and faster DNA sequencing technologies, assembly methods have greatly changed. Instead of outputting reads that are thousands of base pairs long, new sequencers parallelize the task by producing read lengths between 35 and 400 base pairs. Reconstructing an organism's genome from these millions of reads is a computationally expensive task. Our algorithm solves this problem by organizing and indexing the reads using *n*-grams, which are short, fixed-length DNA sequences of length *n*. These *n-grams* are used to efficiently locate putative read joins, thereby eliminating the need to perform an exhaustive search over all possible read pairs. Our goal was develop a novel *n*-gram method for the assembly of genomes from next-generation sequencers. Specifically, a probabilistic, iterative approach was utilized to determine the most likely reads to join through development of a new metric that models the probability of any two arbitrary reads being joined together. Tests were run using simulated short read data based on randomly created genomes ranging in lengths from 10,000 to 100,000 nucleotides with 16 to 20x coverage. We were able to successfully re-assemble entire genomes up to 100,000 nucleotides in length.

Chapter 1

Introduction

The goal of this project is to develop a novel method for the assembly of short read data generated from Next-Generation Sequencers (NGS). Chapter 1 discusses the importance of bioinformatics and describes the recent explosion of biological data. Chapter 2 provides an explanatory background on DNA biology and the advancements in DNA sequencing technologies. Also, the FASTQ file format and previous methods are discussed. Chapter 3 describes the methods and implementations used in the assembler. Chapter 4 details results and discusses metrics used analyzing an assembly. Chapter 5 discusses possible future work and conclusion.

1.1 Introduction to Bioinformatics

In this day and age, computers are an integral part of nearly every aspect of life. Computers allow us to simplify tasks and solve difficult and time-consuming

problems. One field in which computers are playing an increasing role is biology. *Bioinformatics* is the application of statistics and computer science to the field of molecular biology. Computers now are being used to collect, archive, organize, and interpret biological data. Due to the efforts of bioinformaticians worldwide, great strides are being made toward developing an understanding of the biological functions on a level that we have not ever seen.

There is an overwhelming amount of biological data. Organizations, such as the National Center for Biotechnology Information (NCBI), maintain and house genome-sequencing data from thousands of organisms. However, the difficulty lies in trying to interpret and understand the ever-increasing amount of biological information. In response to this influx of data, a large number of tools have been developed to aid researchers in accomplishing these tasks, such as locating genes in a newly discovered genome, understanding various interactions between cells, and understanding how a protein folds (Baxevanis, 2004). One area in particular that has received significant attention is sequence assembly.

1.2 Sequence Assembly

Sequence assembly is a specific category of bioinformatics that pertains to the alignment and reassembly of DNA fragments (M. Pop, Salzberg, & Shumway,

2002). Typically this is seen in DNA sequencing where genome sequencers cannot read the entire genome. Instead, current sequencing technologies split the genome into millions of *reads*, short fragments of DNA, to speed up the task (Section 2.3). The goal is then to use a computer program to reassemble the separated genome. However, these newer technologies have resulted in an enormous increase in sequenced and unassembled genomes. As of March 5, 2012, the NCBI non-redundant sequence database contained 16,923 completed DNA sequences ("NCBI Reference Sequence (RefSeq)," 2012; Pruitt, Tatusova, Klimke, & Maglott, 2009). Even more astonishing is the exponential increase with which the genomes are being sequenced (Figure 1). This inequality results in a great disparity between the number of genomes sequenced and genomes that have been assembled an analyzed. Therefore, the use of computers is essential in reassembling and examining the enormous amounts of sequenced data.

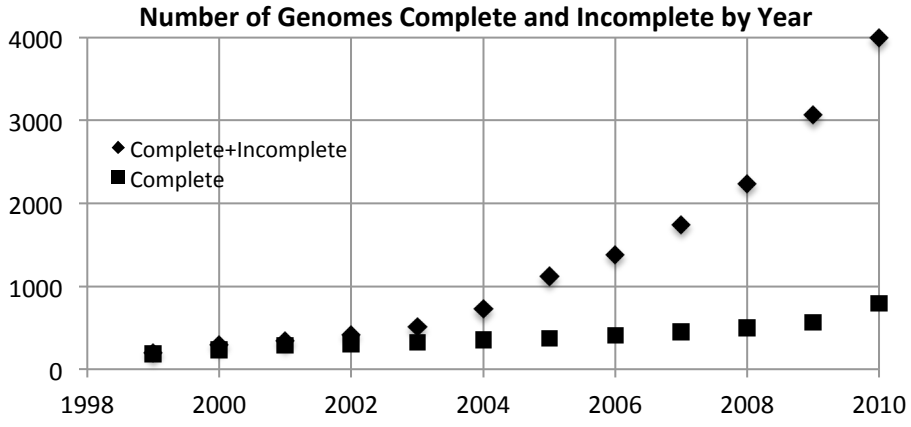

Figure 1 Number of genomes in RefSeq by year

Besides increasing the amount of data, one major advantage to newer sequencing technologies is the rapidly decreasing costs of sequencing new genomes (Figure 2). Eventually, whole-genome sequencing will fall to below $1,000. The $1,000 genome has long been considered the tipping-point for personalized medicine (Davies, 2010). The idea is that once the price drops below that amount, it will finally be cost effective enough to allow doctors to deliver treatment based on a patient's genetic makeup. Instead of administering treatments based on tests and symptoms, a physician will now be able to study a patient's genome to make diagnoses and perfect treatments in everything from diabetes to Alzheimer's.

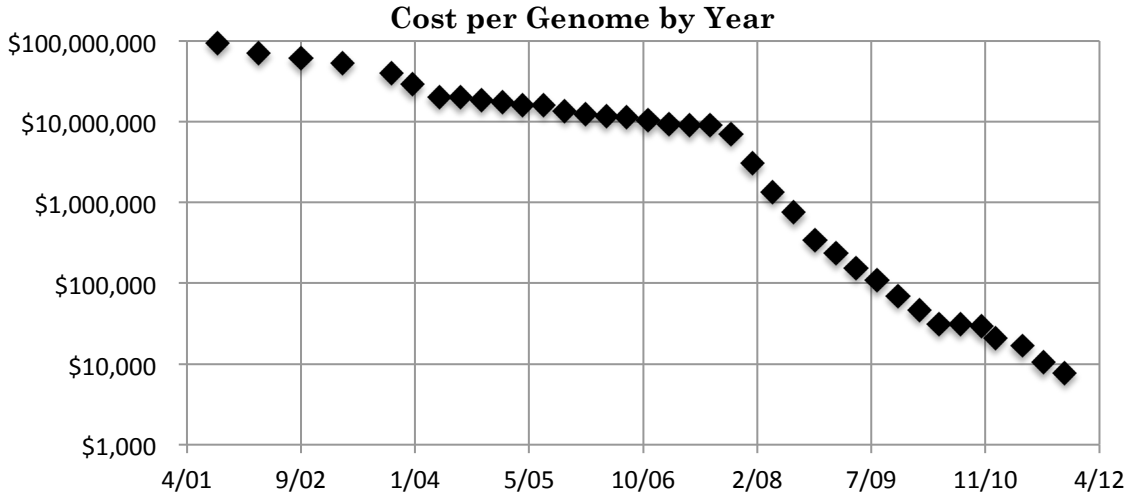

Figure 2 Cost per genome by year

1.3 Chapter Summary

In this chapter, the importance of bioinformatics and its applications was discussed. Additionally, the importance of sequence assembly was emphasized. Newer

technologies have greatly increased the amount of biological data generated. It is the goal of advanced computer programs to reassemble this data into the original genome. Eventually, whole-genome sequencing will be cheap enough for physicians to treat patients based on their genome sequence and prescribe specific treatments based on that information.

In the next chapter, the specifics of DNA sequencing are examined.

Chapter 2

DNA Sequencing and Assembly

The Deoxyribonucleic Acid (DNA) sequence represents the blueprint of every living organism. Understanding this sequence has been crucial in biological research. Ever since Watson and Crick first correctly presented the structure of DNA in 1953, much work has been focused on determining the order of nucleotide bases. However, it wasn't until 1975 that Frederick Sanger developed a method that was fast enough to enable researchers to finally sequence small genomes (Section 2.2). Ever since then, there has been much research on accelerating the sequencing process. Knowledge of the DNA sequence allows researchers to search for regulatory and gene sequences and subsequently compare sequences between species. Sequence comparisons also aid researchers in identifying mutations in unhealthy cells and cancers.

In this chapter, a biological review of DNA will be discussed. The majority of the chapter presents the technologies used in the Sanger method of DNA

sequencing and the more modern Next-Generation Sequencing (NGS) models (Section 2.3). Finally, an overview of the FASTQ file format (Section 2.4) and other assembly methods (Section 2.5) will be examined.

2.1 Biology Background

In all known living organisms, deoxyribonucleic acid (DNA) is the basic genetic material found in all cells. DNA, often referred to the "blueprint of life," carries the instructions for making all the materials and structures needed for a cell to survive. The structure of DNA contains two parts: a backbone and a nitrogenous base. The backbone of a DNA molecule contains alternating phosphate and sugar residues. Attached to the backbone can be one of four nitrogenous bases or nucleobases. Since the backbone of DNA is the same, it is the specific nucleobase that characterizes each nucleotide. The four bases found in DNA are adenine (abbreviated A), cytosine (C), guanine (G), and thymine (T). Furthermore, nucleobases can be classified into one of two types: *purines*, A and G, and *pyrimidines*, T and C (Freeman, 2011).

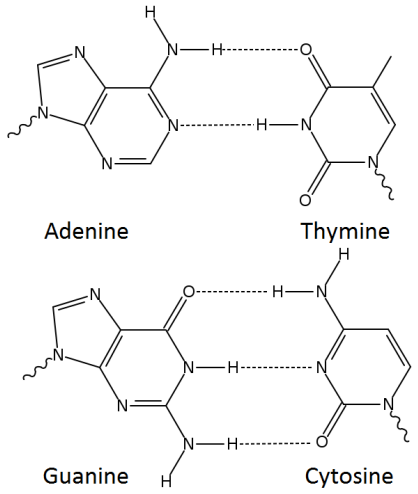

Figure 3 The two base pairs of DNA

DNA is a double helix, forming a unique, stable structure where each nucleobase on one strand interacts with an appropriate base on the other. Also called *base pairing*, one purine molecule hydrogen bonds to one pyrimidine. Furthermore, as seen in Figure 3, an A will only bind to a T and a C will only bind to G (Isilanes, 2007). As stated earlier, each nucleotide is represented using a unique letter. However, in DNA sequencing, ambiguities can arise. For example, it's possible for a nucleotide to be G or a C in a sequence. Therefore, the International Union of Pure and Applied Chemistry (IUPAC) developed abbreviations for different permutations of nucleic acids as specified in Table 1 below.

Table 1 Summary of nucleotide single-letter abbreviations

Symbol	Meaning	Origin of Designation
G	G	Guanine
A	A	Adenine
T	T	Thymine
C	C	Cytosine
R	G or A	puRine
Y	T or C	pYrimidine
M	A or C	aMino
K	G or T	Keto
S	G or C	Strong interaction (3 H bonds)
W	A or T	Weak interaction (2 H bonds)
H	A or C or T	not-G, H follows G
B	G or T or C	not-A, B follows A
V	G or C or A	not-T, V follows U
D	G or A or T	not-C, D follows C
N	G, A, T, or C	aNy

2.2 The Sanger Method

In 1975, Frederick Sanger developed a chain-terminated method that quickly became the method of choice for DNA sequencing. First, the DNA strands are denatured, or separated, using heat. The solution is then divided into four tubes corresponding to the four nucleotides. Next, dideoxynucleotides (ddNTP) are added in addition to the normal nucleotides (NTP). Dideoxynucleotides are similar to normal nucleotides except they prevent the addition of further nucleotides (hence the name chain-terminated method). As the base-strand of DNA is synthesized, normal nucleotides are added to the complementary strand. However, on occasion, a ddNTP is added instead of a normal nucleotide that prevents the further growth of

the strand. This results in many reads of various lengths. For example, strands terminated with a G-dideoxynucleotide (ddGTP) may result in reads as seen in Figure 4.

ATG
ATGCTTCG
ATGCTTCGG
ATGCTTCGGAAG

Figure 4 Potential reads produced from the addition of ddGTP

The process of adding ddNTP and NTP is repeated for the other respective tubes. This results in the whole DNA sequence being terminated at each nucleotide. As seen in Figure 5, each tube is run in separate lanes on a polyacrylamide gel in order to separate each strand by size (Lakdawalla, 2007). The lanes are then merged and read using a computer. Since each chain-terminated dye fluoresces at a different wavelength, a laser can be used to identify each nucleotide in the gel.

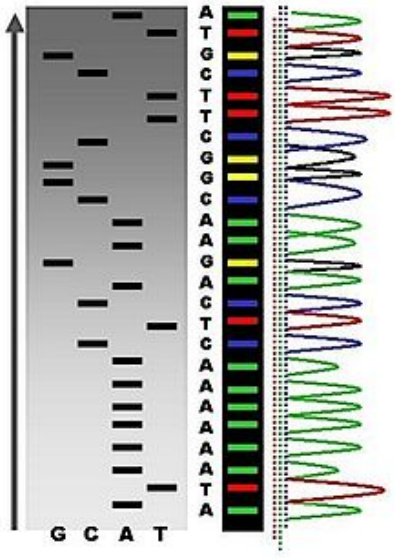

Figure 5 Gel electrophoresis and corresponding wavelengths

2.3 Next-Generation Sequencing

The advent of new sequencing technologies has radically lowered the cost of sequencing large-scale genomes (Section 1.2). Next-Generation sequencing technologies from 454 LifeSciences/Roche and Solexa/Illumina revolutionized the methodologies used in whole-genome sequencing. Next-Generation Sequencers (NGS) parallelize the sequencing task by splitting the copies of DNA into millions of smaller reads and sequencing them simultaneously (Schatz, Delcher, & Salzberg, 2010).

First, the DNA sample is sheared into a collection of fragments using acoustic waves (Mardis, 2008). Special proprietary adapters are then added to the ends of the DNA fragments to aid in binding to the flow cell surface. The flow cell is a specialized surface that aids the binding of DNA and enzymes to the cell surface for manipulation. The fragments are then size separated to determine the optimal fragment length for cloning. Next, the chosen fragments are used as a template strand and cloned using polymerase chain reaction (PCR). PCR is a technique used to drastically amplify the number of copies of a certain region of DNA. The fragments are washed over the flow cell, which then bind randomly to the surface. Subsequently, the DNA undergoes another cloning step to form clusters of the same fragment on the flow cell surface. Next, the four bases are added one at a time to

the flow cell. Since each base emits a certain color dye, a camera can be used to image the flow cell after each round as seen in Figure 6 (Mardis, 2008).

Finally, from this image, the DNA sequence can be generated for thousands of DNA fragments at the same time resulting in a collection of reads – DNA sequences whose sequence is known (Mihai Pop, 2009). This is accomplished by using a computer to analyze certain pixels in the image. Since the location of the DNA reads does not change, a change in pixel color at a specific location corresponds to a different nucleotide in the sequence. For example, as indicated in Figure 6, the yellow 'G' in the first slide changes to a blue 'C' in the next.

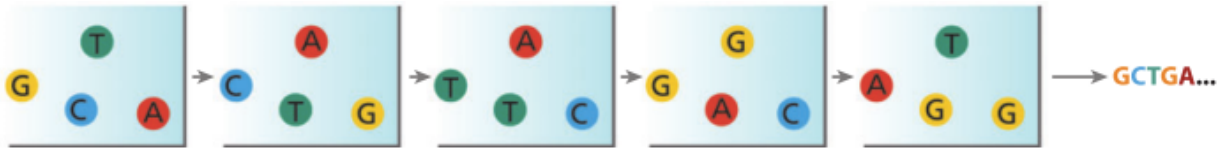

Figure 6 Sequence read over many chemistry cycles

2.4 The FASTQ Format

In DNA sequencing, the FASTQ file format has emerged as the *de facto* format for sharing NGS data between tools (Cock, Fields, Goto, Heuer, & Rice, 2010). Sequencing instruments such as Roche/454 and Illumina/Solexa use the FASTQ format for storing their outputted data. Developed by the Wellcome Trust Sanger Institute, the FASTQ format combines the FASTA sequence and the corresponding quality data. The file format normally uses four lines per read and encodes the

nucleotide and quality value with a single ASCII character (Figure 7). The first line begins with a '@' character and only contains a header and optional sequence identifier. Line 2 is the nucleotide sequence of the DNA read outputted from the sequencer. The third line begins with a '+' character and, similar to Line 1, only contains a header and optional sequence identification. Finally, Line 4 contains the quality values for the corresponding nucleotides in Line 2.

```
@SRR090119.2 length=25
TTACAAGTCAAAGCCCTAACCGGTA
+SRR090119.2 length=25
@8;==4<<<EA3<?;$<A9?7B:=@
```

Figure 7 Example of a minimal FASTQ file

Typically, software tools such as PHRED, the industry standard nucleotide identification software package, are used to determine the nucleotide and corresponding quality value (Ewing & Green, 1998). The quality value, also called the PHRED score, is an integer indicating the likelihood that the sequencer generated the correct nucleotide. By definition, the quality value QV is derived from the probability P that the base call is incorrect as defined in (Cock et al., 2010):

$$QV_{PHRED} = -10 * log_{10}(P)$$

For example, a quality score of 30 would correspond to a base call accuracy of 99.9%.

Each machine technology encodes the quality value using their own scoring mechanism. A standard approach to address this non-uniformity is to standardize the score to follow the original PHRED standard. The Sanger FASTQ files use the ASCII values 33-126 to encode PHRED scores of 0-93. This corresponds to a PHRED score with an ASCII offset of 33. Solexa sequencing uses a logarithmic mapping with scores ranging from -5 to 62. The equation to convert the Solexa quality value into PHRED is:

$$QV_{Solexa} = 10 * log_{10}(10^{\frac{QV_{PHRED}}{10}} - 1)$$

Illumina sequencing initially used the Solexa format but eventually switched to a third FASTQ variant (Cock et al., 2010). The Illumina FASTQ encode PHRED scores ranging from 0-62 and corresponds to an ASCII score of 64-126 (i.e. an offset of 64). A summary of the three FASTQ variants is described in the table below (Cock et al., 2010).

Table 2 Summary of the three described FASTQ variants

Description, name	ASCII characters		Quality scores	
	Range	Offset	Type	Range
Sanger standard `fastq-sanger`	33-126	33	PHRED	0 to 93
Solexa `fastq-solexa`	59-126	64	Solexa	-5 to 62
Illumina `fastq-illumina`	64-126	64	PHRED	0 to 62

2.5 Assembly Methods

By definition, an *assembly* is a hierarchical data structure that maps the sequence data to the putative reconstruction of the target (Miller, Koren, & Sutton, 2010). Since a genome is sequenced in only small portions, a software program can be used to reassemble the complete genome. Software assemblers focus on one obvious assumption: if two sequence reads share a common overlapping substring of letters, then it is because they *are likely to* have originated from the same chromosomal region in the genome (Narzisi & Mishra, 2011).

The general algorithm behind sequence assembly is straightforward. A large number of candidate pairs of reads are tested to see if they can be joined based on the assessment of various characteristics of the overlapping region between the reads. A scoring mechanism is used to quantitatively compare different putative joined reads. The algorithm picks the highest scoring overlap before merging the two reads. The process is repeated until no more moves/merges can be made.

The problem of sequence assembly is similar to the problem of finding the shortest common supersequence (SCS), a well-known problem in computer science (Cormen et al., 2009). The SCS is defined as a common superstring of minimum length between two given sequences. In other words, sequence **C** is a supersequence of subsequences **A** and **B** if the shortest sequence **C** contains both subsquences **A**

and **B.** For example, given two sequences **A** = ACGCTAC and **B** = CTGACA, the SCS of **A** and **B** is: **C** = ACGCTGACA as indicated in Figure 8. Therefore, sequence **C** is the shortest sequence that contains both **A** and **B**.

A = ACGCT AC
B = C TGACA
C = ACGCTGACA

Figure 8 Shortest common supersequence of two reads

It is important to note that there exists two approaches for the assembly of genomes: *de novo* and reference-based. *De novo* methods are aimed at reconstructing genomes that are not similar to any organism previously sequenced (Mihai Pop, 2009). Conversely, reference-based assemblers use the sequence of a closely related organism to aid the assembly process. This thesis project focuses on *de novo* assemblies, however, for sake of completeness, both method backgrounds are briefly discussed.

2.5.1 Reference-based Assembly

Reference-based assembly, also called mapped assembly, replaces overlap detection with alignment against a similar genome. Similar to the shortest common subsequence problem, an alignment is a way of rearranging and comparing DNA to identify commonalities between the sequences (Baxevanis, 2004). An alignment

increases the number of matches between the sequences in order to increase the similarities between them. This is accomplished by taking the DNA sequences and adding gaps to one or both sequences in order to maximize the similarities between the reads. Computationally, reference-based assemblies are much faster since it eliminates the need for determining overlaps (Mihai Pop, 2009). First, the sequenced reads are aligned to the reference sequence under the assumption that the reference is similar to the newly sequenced genome. Next, the alignment is used to compute a consensus sequence of the new genome. However, this approach is error-prone if the reference is vastly different than the sequenced reads ("NCBI: Assembly Basics," 2012).

2.5.2 *De novo* Assembly

The method of analyzing and determining the evaluation of overlaps for *de novo* assemblers falls into two main categories: *greedy* and *graph-based*. A *greedy* algorithm is an algorithm that makes the optimal choice at each stage of the problem solving process. Even though a greedy algorithm is fairly easy to implement, a major flaw is that it focuses too much on the current stage and ignores long-term optimal joins. Furthermore, the memory requirements to implement such algorithms are expensive. Current implementations require up to one gibabyte of RAM for each megabase of assembled sequence (M. Pop et al., 2002). For both of

these reasons, the effectiveness of greedy algorithms is limited. Currently, TIGR, CAP3, and PHUSION are well-known assemblers in this category (Huang & Madan, 1999; Mullikin & Ning, 2003; G. G. Sutton, White, Adams, & Kerlavage, 1995).

To combat this problem, a *graph-based* approach to sequence assembly was developed. Instead of focusing on the raw sequences, graph-based algorithms start by preprocessing the sequence to determine overlap information to store in an unweighted edge-based string-graph (Narzisi & Mishra, 2011). The algorithm first starts by dividing the sequence into a collection of n-grams, which are short, fixed-length DNA sequences of length n. Next, a de Bruijn graph is constructed. A de Bruijn graph is a directed graph where each edge represents a fixed-length overlap and each node corresponds to overlaps of n-1 bases (Miller et al., 2010). Finally, an Eulerian path is found containing each edge exactly once (Pevzner, Tang, & Waterman, 2001). The result is a genome sequence consistent with the n-gram data on every read (Figure 9). Graph-based algorithms benefit greatly from their speed. An Eulerian path, in theory, can be calculated in $O(n)$ time.

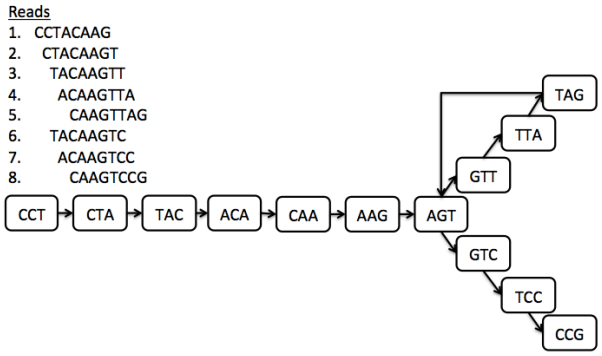

Figure 9 Set of 8 8-bp reads and corresponding de Bruijn graph

In reality, three main factors cause complications in an assembly. First, sequencing errors in the read data can cause false-positive joins and mislead the algorithm (Narzisi & Mishra, 2011). Most assemblers implement an error-correcting stage to account for such problems. Second, double stranded DNA and palindromes can cause paths to repeat themselves continuously. Third, and most importantly, repeat structures cause much of the complexity associated in determining optimal assemblies. If a repeat is longer than the *n*-gram, then the graph will have the same *n*-gram corresponding to multiple locations on the genome. The algorithm does not contain enough information to disambiguate the repeat (Miller et al., 2010). Current examples of de Bruijn graph-based assemblers include Velvet, ABySS, and SOAPdenovo (R. Li et al., 2010; Simpson et al., 2009; Zerbino & Birney, 2008).

Although all three assemblers utilize a de Bruijn graph, the implementation and methodologies for each program is different. Velvet makes extensive use of a graph simplification to reduce non-intersecting paths to single nodes (Miller et al.,

2010). Simplification compresses the graph without losing essential information. Velvet also has a parameter for the minimum number of *n*-gram occurrences needed to be considered a graph node. This step eliminates *n*-grams that are highly probable of being error-prone. Velvet utilizes a breadth-first-search to first scan to the perimeters of the graph and removes reads below a certain threshold (Zerbino & Birney, 2008). Finally, after other error-avoidance steps, Velvet forms contigs by traversing the graph similar to Figure 9.

ABySS differentiates itself by distributing the *n*-gram graph across a compute grid whose combined memory is typically larger than a traditional desktop computer (Miller et al., 2010; Simpson et al., 2009). Typically suited for Illumina/Solexa data sets, AByss assigns each node in the graph to a specific CPU in the cluster. After running through error-correcting steps, the graph is traversed to form contigs. However, the traversal is not performed on one CPU. Each node stores information on the successor node in the traversal. By switching CPUs, the algorithm offsets the requesting and retrieval of information to other CPUs (Zerbino & Birney, 2008).

SOAPdenovo is one of the few freely available assemblers capable of assembling mammalian genome sequences of Illumina/Solexa reads (Miller et al., 2010). SOAPdenovo utilizes a rigorous error-correcting step to remove any possible error-prone reads. This saves a significant amount of space, but at the cost of

potentially removing useful read information. Finally, a traversal on the de Bruijn graph is performed similarly to the methods described above.

2.6 The Need for Accurate Assemblies

Even though there are many NGS assemblers currently available, there is no commonly accepted and standardized method for the task. Additionally, validating if an assembler outputs the correct sequence is a difficult and time-consuming task. The most widely accepted metric for evaluating an assembly is the N50 scoring metric. The N50 is defined as the largest number L such that the combined length of all contigs of length $\geq L$ is at least 50% of the total length of all contigs (Narzisi & Mishra, 2011). An example of an N50 calculation is given in Figure 10.

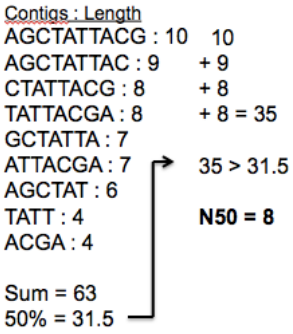

Figure 10 A sample N50 calculation using 9 contigs

Moreover, the current metrics, whether it is the N50 or contig size, only emphasize size and not the quality of the joins. In this case, an assembler can sacrifice optimal

joins for longer contigs to give the appearance of outperforming others. There is thus an expressed interest in the bioinformatics community for an assembler that takes into consideration the quality of the joins to form an optimal assembly. This thesis research hopes to address this problem by developing a novel *n*-gram method for the assembly of genomes from next-generation sequencers. Specifically, a probabilistic, iterative approach will be utilized to determine the most likely reads to join through development of a new metric that models the probability of any two arbitrary reads being joined together.

2.7 Chapter Summary

In this chapter, the necessary background information of sequence assembly and DNA was discussed. For the computer scientist, an introduction to the biology of DNA and the rules regulating base pairing was presented. DNA sequencing, from the original Sanger method to today's current next-generation sequencing, has greatly transformed the way researchers analyze biological data. The FASTQ file format and its importance was also explained. Finally, an analysis of previous methods and their limitations was examined.

In the next chapter, the specific methods and implementations of the research program will be discussed.

Chapter 3

Methods and Implementation

The functionality of the program can be broken down into two distinct parts. First, after being filtered through a data cleaning step, raw sequence reads are read in to form a dictionary of *n*-grams. Second, the necessary assembly is made using a novel method to determine the optimal joining of two arbitrary reads. The assembly also requires a bookkeeping step to update the dictionary on the newly formed contigs –a set of overlapping DNA segments that together represent a consensus region of DNA.

3.1 Pre-Processing

In data mining, pre-processing is an essential step that precedes data analysis. Pre-processing allows for the screening of data points to remove out-of-range values, impossible data combinations, or missing values (Kotsiantis, Kanellopoulos, & Pintelas, 2006). In sequence assembly, data pre-processing is used to filter and

remove low quality reads. Specifically, three standards are used to remove low quality reads from the data set and prevent them from being added to the dictionary. First, the read must be longer than the *n*-gram length. An *n*-gram is a subsequence of DNA of length *n* and forms a critical basis for indexing all of the reads, and thus represents the minimal length read allowed. For example, if the read is 18 characters long but the *n*-gram length is 22 then the read is too short to be useful in joining. Second, an entry can be filtered based on the average quality of the read. A read with a low average QV score does not contain nucleotides with a high probability of being correct. Thus, low QV scores can result in incorrect joins. Finally, an entry can be filtered on the percentage of N's in a read. As seen in Table 1, the IUPAC single-letter abbreviation N appearing in the sequence means that position in the sequence can be any of the four nucleotides. Therefore, there is no information on what the nucleotide should be at that particular location. Again, a high percentage of N's in a read can result in low quality and improper joins. Setting a threshold for both the average QV score and percentage of N's in a read can eliminate reads that do not contain a high probability of being correct.

The benefits of data pre-processing are two-fold. First, the size of the dictionary is minimized. Eliminating low quality data entries in an analysis greatly speeds up the processing time. In the case of sequence assembly, reducing the number of entries in the dictionary, without sacrificing quality, can greatly

minimize the number of comparisons needed to determine an optimal join. Finally, pre-processing limits the data to include only high quality entries. Removing incorrect reads from the dictionary decreases the probability of forming an incorrect join.

3.2 *n*-gram Dictionary

Before we start an assembly, the initial set of sequence reads must be read and stored in the system. This is accomplished using FASTQ files (Section 2.4). The next, and arguably most crucial step, is to map reads to sequences of origin (Horner et al., 2010). The main data structure to store this information is the C++ Standard Library `map`. Maps are a type of associative container that store information based on a *key* and *mapped* value. In this instance, a map is used to match an *n*-gram to a vector of read indices. A secondary data structure, called `readIndices`, is used to store three integers: a read index, indicating the read identifier the *n*-gram was found in, the starting index of the *n*-gram in the read, and the distance the *n*-gram is located from the end. Therefore, each `readIndices` represents each instance of the *n*-gram in the short-read data set. An example of a dictionary is given in Figure 11.

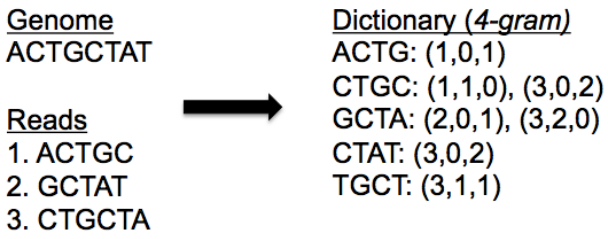

Figure 11 Example of reads and the corresponding dictionary

The structure `readIndices` also allows for a quick way of calculating where in a read the *n*-gram resides. Another data structure is required to store the reads. A vector of reads allows for a quick lookup of the specific read sequence and quality value. The index for each read corresponds to the read index stored in the `map`. The dictionary creation method is outlined in Figure 12.

```
1.  read in FASTQ file
2.  start = 0
3.  end = ngramSize
4.  while a line exists
5.    read in seq and qv
6.      create vector of readfragIndices
7.    while end is less than read length
8.      ngram is the substring of read from start to end
9.        readIndices ← start, end, and index
10.       add readIndices to vector
11.       increment start and end
12.       add sequence and qv to reads vector
```

Figure 12 Outline of the dictionary creation method

3.3 Assembly

An *assembly* is a hierarchical data structure that maps the reads from the NGS instrumentation together in such a way that a putative reconstruction of the target

is generated (Miller et al., 2010). The assembly function plays the largest role in the genome reconstruction and is outlined in Figure 13. The main loop of the function iterates through every *n*-gram in the dictionary. For each *n*-gram, three functions are executed. First, a probabilistic scoring model is used to determine the optimal joins between two reads that contain the specific *n*-gram. Second, the actual assembly is made based on the scoring model. Finally, bookkeeping is performed to update the dictionary. Each of the three functions is explained in more detail below.

```
mainAssembly() {
      for each n-gram in dictionary {
            find best join from sequences containing n-gram;
            assemble the join;
            update data structures
      }
}
```

Figure 13 Top-level outline of main assembly function

3.3.1 Notation

First, we define the notation required to formalize the explanation of the method. Let G represent an arbitrary sequence of DNA to be processed by some NGS instrumentation. Let ***D*** represent the set of all reads that were output by the NGS instrumentation that processed G. In an ideal world, G is observable information, and we simply map the reads back to G. However, in this problem, G is hidden, unobservable information. The aim of a *de novo* assembly method is to determine

the most likely assembly of sequences in $\boldsymbol{D}$ that give us back the original sequence G.

Let X_i represent the i^{th} read in $\boldsymbol{D}$ of length m. Let s represent an arbitrary nucleotide in sequence X, then $X_i = s_1,s_2,\ldots,s_m$, and X_i has m-n+1 n-grams. Let ng_k represent the k^{th} n-gram in the data. Every n-gram ng_k forms a key for referencing every X_i that contains ng_k in its sequence, and serves a critical role by providing efficient assessment of the quality of the overlapping region between X_i and X_j, and serves as a key to mapping an index indicating the precise location of that n-gram in the data.

3.3.2 Determining Optimal Joins

For our work, we define an optimal join over all possible pairs of reads in $\boldsymbol{D}$ as being the join that is most likely to come from the same origin. Considering all possible joins is $O(|\boldsymbol{D}|^2)$, multiplied by the cost of performing an alignment of both sequences being considered. Considering that typical datasets can contain millions of reads, and that the vast majority of these joins should not be considered, a mechanism is needed to efficiently reduce the search space of possible joins. The n-gram index serves to substantially reduce the reads to consider for joining to those that share a common n-gram.

Aligning two sequences is similar to the problem of finding the longest common subsequence between two sequences (Section 2.5). In a true alignment, we allow a limited number of insertions and deletions (indels) in one or both of the sequences in order to maximize the number of matching symbols between them. However, in this work, we do not consider insertions and deletions due to the low likelihood that these errors are introduced by the sequencing instrumentation. This also avoids the computational complexity that arises if indels were considered. (Modeling this error is left for investigation in future work.) For two sequences of length m_1 and m_2, the cost of aligning two sequences is $O(m_1 m_2)$.

Every n-gram ng in $\boldsymbol{D}$ has a set of reads $\boldsymbol{D}_{ng} = \{X_1 \dots X_k\}$ that contain ng in the sequence. For each combination of reads, a score S is calculated. For example, $S_{1,2}$ is the score resulting from joining reads X_1 and X_2. More specifically, score S is the sum of the calculations from three separate regions in the resulting joined sequence. The reads are arranged such that they follow a strict pattern shown in Figure 14. Given two indices i and j for reads in $\boldsymbol{D}_{ng}$, the reads are arranged such that region A is found only on X_i while region C is found only on X_j. Region B is overlap of X_i and X_j and is thus found in both reads. Each n-gram ng will thus have a $|\boldsymbol{D}_{ng}|$ by $|\boldsymbol{D}_{ng}|$ matrix created.

Region:

	A	B	C
X_i =	ACGT	CGATTA	
X_j =		CGATTA	CGATAC

Figure 14 Arrangement of two reads for assembly

The score for each potential join is computed by the function *scoreJoin*(X_i, X_j). This function uses a simple probabilistic model to assess the likelihood that these two reads came from the same origin in G. The resulting score is used to judge this join against all other possible joins being considered for this *n*-gram. More specifically, this score represents the probability that the two reads being considered are joined together by random chance. The lower the probability that these two sequences came from the same origin by chance, the more likely that these two sequences originated from the same place in G.

To obtain a probability for the join, we first assume that each read is generated according to a random process, and is identically and independently distributed. Each position in the sequence has a probability of taking on one of four nucleotides. We make two important assumptions for simplicity. First, we assume a uniform distribution of nucleotides for any position, implying that each nucleotide is equally likely. Therefore, $P(\{A, C, G, T\}) = {}^1/_4$. The symbol N in the sequence implies that the position could be any nucleotide, $P(\{N\}) = 1$. Second, we make a Markov

assumption, which assumes that the occurrence of each nucleotide in the sequence is independent of the previous nucleotide.

The score for the join is broken into scoring three distinct regions, denoted as *A*, *B*, and *C* (Figure 14). Regions *A* and *C* use a similar calculation. Since the confidence of a nucleotide decreases the farther away the nucleotide is from the overlapped region (region *B*), a discount factor equation was created to show the degradation in nucleotide confidence. Therefore, the farther away from *B*, the less confidence, and thus lower probability, the nucleotide is of being correct. Given α to be the probability of a non-overlapped nucleotide, β to be the discount factor, and d being the absolute distance from region *B*, the probability of a nucleotide N at position i is shown in Equation 3.1. Typically, α is $1 - P(nucleotide) = 1 - 0.25 = 0.75$.

$$P(N_i) = \alpha\beta^d \tag{3.1}$$

Therefore, the probability for region *A* is the sum of the probabilities of the individual nucleotides in region *A*. Similarly, the probability for region *C* can be calculated by summing the probabilities of the individual nucleotides in *C*. We compute log probabilities instead of standard probabilities to simplify the computation. Using log probabilities has two main advantages. First, the calculation speed is increased. Multiplying probabilities is much slower computationally than addition. Second, log probabilities aid in number accuracy.

Multiplying small probabilities over a long sequence can result in an underflow error – a small number outside the precision limits of a computer. Therefore, log probabilities prevent these problems without compromising the accuracy of the final answer.

Instead of taking into consideration the probability of a single nucleotide, region *B* requires the probability of two nucleotides. In most alignment methods where mismatched symbols might need to be aligned, a substitution matrix is often used to make the computation of the final score for the alignment more efficient. There are five symbols that we consider as indicated in Table 3.

Table 3 Substitution matrix for sequence alignment

	A	**C**	**G**	**T**	**N**
A	$\left(\frac{1}{4}\right)^2$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{1}{2}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\left(\frac{1}{4}\right)$
C	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\left(\frac{1}{4}\right)^2$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{1}{2}\right)$	$\left(\frac{1}{4}\right)$
G	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{1}{2}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\left(\frac{1}{4}\right)^2$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\left(\frac{1}{4}\right)$
T	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{1}{2}\right)$	$\binom{2}{1}\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)$	$\left(\frac{1}{4}\right)^2$	$\left(\frac{1}{4}\right)$
N	$\left(\frac{1}{4}\right)$	$\left(\frac{1}{4}\right)$	$\left(\frac{1}{4}\right)$	$\left(\frac{1}{4}\right)$	1

For example, consider a sequence X_1 = GCTATAA and X_2 = CTGCTACN. Suppose both of these are considered for the n-gram GCTA. In this case, X_2 would be the left sequence in the join, and X_1 would be the right as indicated in Figure 15.

Region:

	A	B	C
X_i =	CT	GCTACN	
X_j =		GCTATA	A

Figure 15 Example of sequence alignment and score calculation

Regions *A* and *C* are calculated as follows:

$$A = -\log(0.75 * 1.001^2) - \log(0.75 * 1.001^1) = 0.2486$$
$$C = -\log(0.75 * 1.001^1) = 0.1241$$

Region *B* requires the substitution matrix and a nucleotide-to-nucleotide comparison.

$$G - G = -\log(0.25 * 0.25) = 1.2041$$
$$C - C = -\log(0.25 * 0.25) = 1.2041$$
$$T - T = -\log(0.25 * 0.25) = 1.2041$$
$$A - A = -\log(0.25 * 0.25) = 1.2041$$
$$C - T = -\log(({}_2C_1) * 0.25 * 0.50) = 0.9031$$
$$N - A = -\log(0.25) = 0.6021$$
$$Sum = 6.3212$$

Therefore, the score *S* is the sum of the three regions.

$$S = 0.2486 + 6.3212 + 0.1241 = 6.6939$$

3.3.3 Assembly

The assembly function combines the two reads with the highest probability score calculated in the previous section and forms a new contig – a set of overlapping reads that represent a consensus region of DNA. Given two indices *i* and *j* for reads

in $\boldsymbol{D}_{ng}$, let X_i and X_j be the two reads with the highest join score. Similar to the determining optimal joins function, X_i and X_j are separated into three similar regions. The reads are arranged such that region A is found only on X_i while region C is found only on X_j. Region B is the overlap of X_i and X_j and is thus found in both reads. Adding regions A and C to the new contig is straightforward, however region B requires the comparison of two separate read sequences. A character-to-character comparison is made for each nucleotide in B. In Region B, given $X_{i,k}$ and $X_{j,k}$ to be the nucleotide characters at position k for read X_i and X_j, J_k to be the nucleotide at position k at newly formed contig J, and $QV_{X_{i,k}}$ and $QV_{X_{j,k}}$ to be the quality value sequences for X_i and X_j at position k, the comparison rules are:

1. If $X_{i,k} = X_{j,k}$ then $J_k = (X_{i,k} \mid X_{j,k})$
2. If $X_{i,k} \neq X_{j,k}$ then
    a. If $QV_{X_{i,k}} > QV_{X_{j,k}}$ then $J_k = X_{i,k}$
    b. If $QV_{X_{i,k}} < QV_{X_{j,k}}$ then $J_k = X_{j,k}$
3. If $X_{i,k} = N$ and $X_{j,k} \neq N$ then $J_k = X_j$
4. If $X_{i,k} \neq N$ and $X_{j,k} = N$ then $J_k = X_i$
5. If $X_{i,k} = N$ and $X_{j,k} = N$ then $J_k = N$

This can be summarized as follows. If the two nucleotides are the same, add the nucleotide to the contig. If they are different, add the nucleotide with the higher quality value to the contig. If either nucleotide is a 'N', add the other nucleotide. If both nucleotides are 'N', add a 'N' to the contig. After the three regions have been merged, if the contig is unique (i.e. there exists no other contigs with that particular

sequence) then it is added to the `reads` vector. Quality values are joined in a similar manner. In Regions *A* and *C*, the QV is copied over directly. In Region *B*, the average of the two QV scores is added to the contig.

3.3.4 Bookkeeping

The goal of the bookkeeping function is to update the dictionary on the newly formed contig. Bookkeeping is broken down into two separate stages. The first step is to remove the old read entries. This is accomplished by scanning the entire dictionary looking for the `readIndices` of the two reads used in the join. Let *f* and *g* correspond to the location in the `reads` vector for the reads X_i and X_j used in the assembly. Next, every entry in ***D*** is scanned looking for `readIndices` containing *f* and *g*. When either index is found, the `readIndices` is removed from ***D***. Finally, each *n*-gram of the new contig is added to ***D*** similar to the dictionary creation step (Section 3.2). Given *m* being the length of the contig, the number of *n*-grams added is *m*-*n*+1. An example of the Bookkeeping step is shown in Figure 16.

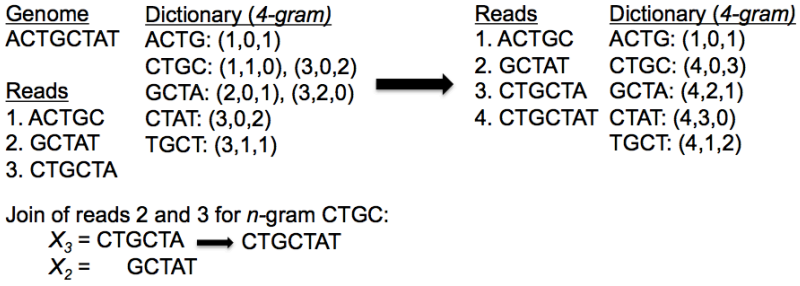

Figure 16 Example of the bookkeeping function

3.4 Chapter Summary

In this chapter, an overview of the two distinct steps in the assembly methods was discussed. First, the dictionary is created by mapping an n-gram to a vector of indices corresponding to the location of each instance of the n-gram. Next, the assembly is made by iteratively traversing the dictionary. First, the optimal join is determined by using a probabilistic method that takes into the consideration the occurrence of the join to chance. Next, the actual assembly is made. Finally, a bookkeeping step is used to update the dictionary on the newly formed contig.

In the next chapter, the results from the implemented assembly methods are shown.

Chapter 4

Results

Analyzing a DNA assembly is a difficult task. Typically, due to errors in DNA sequencing, complete assemblies are not made. Instead, most assemblers output the longest contigs and use a secondary program, like *AMOS*, to merge the contigs together into scaffolds – a portion of DNA reconstructed from contigs and the appropriate gaps ("AMOS," 2010). Therefore, typical assembly metrics are based on the N50 value, a measure of the coverage, and the length of the longest contigs. Analysis of this thesis project is conducted using simulated data.

4.1 Simulated Data

A simulated data set benefits the researcher by being able to compare the assembly results to the known solution. Additionally, since simulated data sets are typically smaller than real genomes, the time needed to perform an assembly is decreased significantly. For this analysis, two simulated data sets were generated from a self-

made helper program. A 10,000 nucleotide genome with 20x coverage and an average read length of 40, hereby called G10k, and a 100,000 nucleotide genome with 16x coverage with an average read length of 50, called G100k, were generated using self-made helper programs. The G10k data set resulted in 8,005 reads while the G100k data set contained 66,636 reads. The QV score was a strict 40 for every nucleotide.

Analysis was conducted by observing the longest contig and execution time. If a complete assembly was not made, the N50 was calculated. A total of 10 trials with varying *n*-gram lengths were run for both G10k and G100k. Figures 17 and 18 show the *n-gram* length and the execution time for G10k and G100k respectively. The G100k trials resulted in 6 of 10 being completely assembled while 5 of 10 G10k trials were completely assembled for the simulated genomes. The results of all 20 trials are shown in Table 4 below.

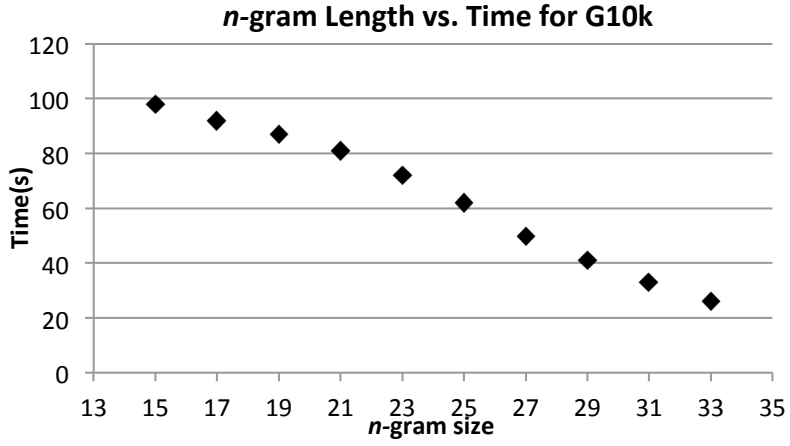

Figure 17 Assembly time of various *n*-gram lengths for G10k

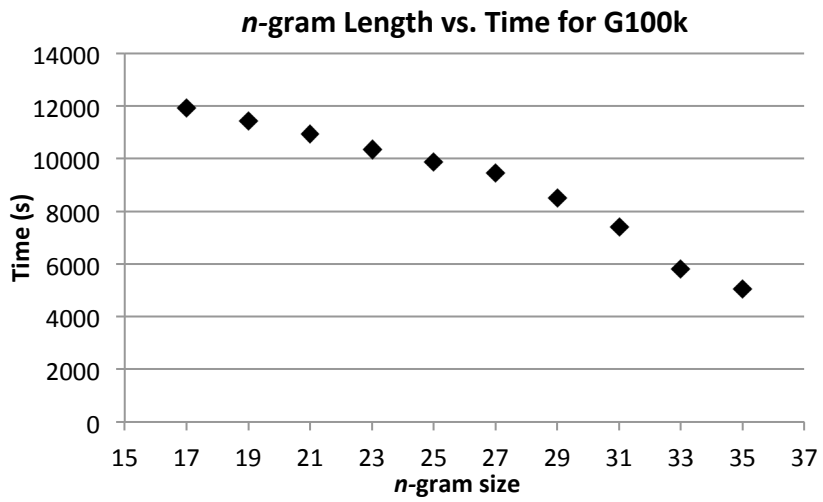

Figure 18 Assembly time of various *n*-gram lengths for G110k

Both Figures 17 and 18 indicate that *n*-gram size greatly impacts the speed of an assembly. If an *n*-gram is too small, the number of occurrences for each *n*-gram increases. This results in a higher execution time to create a matrix and determine the optimal join (Section 3.3.2). Conversely, too large an *n*-gram size results in too many unique *n*-gram occurrences. Consequently, there isn't enough data to support an optimal join of two reads. This can be seen in the larger *n*-gram trials for both G10k and G100k (Figures 19 & 20).

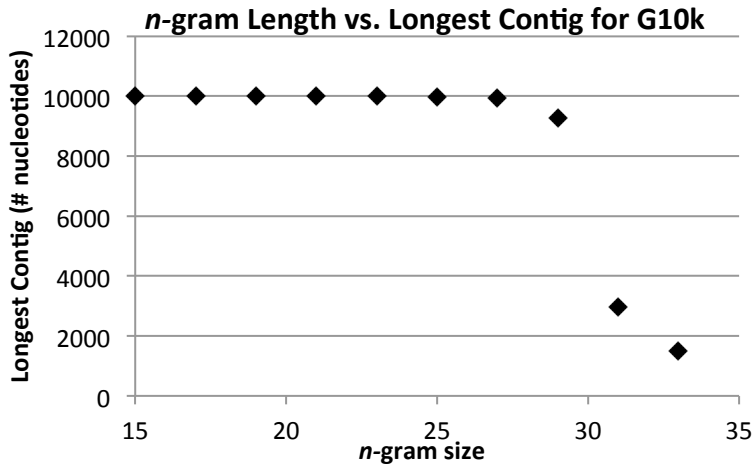

Figure 19 Length of longest contig by various *n*-gram siezes for G10k

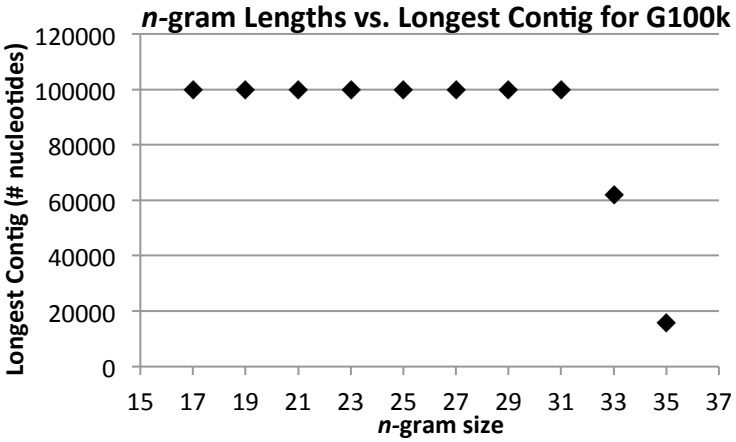

Figure 20 Length of longest contig of various *n*-gram sizes for G110k

Due to the smaller average read length, a large *n*-gram size results in not enough information being known to make an optimal join resulting in a shorter longest contig. Therefore, the *n*-gram size is crucial in obtaining an optimal assembly in both accuracy and speed.

Table 4 Assembly results for G10k and G100k

G10k				G100k			
***n*-gram**	**Time (s)**	**N50**	**Longest Contig**	***n*-gram**	**Time (s)**	**N50**	**Longest Contig**
15	98	188	10000	17	11927	299	100000
17	92	151	10000	19	11402	264	100000
19	87	125	10000	21	10935	255	100000
21	81	105	10000	23	10355	177	100000
23	72	97	10000	25	9879	170	100000
25	62	85	9974	27	9437	147	100000
27	50	82	9949	29	8499	126	99933
29	41	76	9259	31	7393	114	99933
31	33	67	2964	33	5811	98	62002
33	26	62	1499	35	5056	88	15641

4.2 Chapter Summary

In this chapter, experiments were run using simulated datasets for genomes of lengths 10,000 and 100,000 (referred to as G10k and G100k respectively). A total of 20 trial runs of various n-gram lengths were conducted for both G10k and G100k. The G10k tests resulted in 5 of the 10 runs being successfully assembled while the G100k tests resulted in 6 of 10 successfully assembled trials.

In the next chapter, future work and concluding remarks are presented.

Chapter 5

Conclusion and Future Work

This thesis has shown that a probabilistic *n*-gram based method can be used to successfully assemble small length genomes. Using simulated datasets for genomes of lengths 10,000 and 100,000 (referred to as G10k and G100k respectively), 10 trial runs of various *n*-gram lengths were conducted for both G10k and G100k for a total of 20 trials. The G10k tests resulted in 5 of the 10 runs being successfully assembled with a fastest, accurate assembly of 72 seconds. The G100k trials assembled 6 of the 10 runs with a fastest, accurate assembly of 9,437 seconds (2:37). Both datasets exhibited similar behavior with larger *n*-grams. The larger the *n*-gram, the shorter the longest contig assembled. This is caused by too much uniqueness in the dictionary. Too large an *n*-gram size results in too many unique *n*-gram occurrences. Therefore, there isn't enough data to support an optimal join of two reads.

Sequence assembly is a challenging task. Currently, no method, algorithm, or implementation solves the whole-genome assembly problem (Miller et al., 2010). Mathematically, *de novo* genome assembly has been proven to be difficult by being classified in a set of problems (NP-hard) where there is no efficient computational solution (Medvedev, Georgiou, Myers, & Brudno, 2007; E. W. Myers, 1995; Mihai Pop, 2009). However, sequence assembly is still an evolving field. With new research constantly being published, sequencing and assembly technologies are quickly evolving (Chapter 1). Furthermore, there is an expressed interest in accurate DNA assemblies (Section 2.6).

One area in particular that will benefit greatly from improvements in DNA sequencing is healthcare. Being able to quickly and accurately sequence a person's genome will allow doctors to get a precise analysis of predisposed illnesses and determine the best method of treatment for various ailments. Cancer, for example, is notoriously difficult to treat. Recently, improvements in DNA sequencing have shown that cancer is not a homogenous region of the same mutated DNA (Gerlinger et al., 2012). Instead, there are many mutations that are not consistent across the same tumor. Although a momentary setback in cancer treatment research, without the advancements in DNA sequencing, researchers would have been focusing their time and efforts on incorrect information.

DNA sequencing is increasingly becoming an integral role in genomic research. As this thesis research has shown, novel methods for whole-genome sequence assembly can accurately assemble small length genomes. However, much work is still yet to be done. As technology advances, the volume of data generated by high-throughput sequencing will continue to increase. Assembly algorithm developers will have to continue to adapt to the endless challenges associated with *de novo* assemblies. Assembly is not a solved problem, but continued advancements can greatly change the biological research landscape for years to come.

5.1 Future Work

Further optimization of the assembly program can be achieved in many areas. The utilization of paired-ends and scaffolding would greatly increase the size of the assembled genome. In DNA sequencing, the length of the fragments typically exceed the read length achievable by a sequencing technology (Mihai Pop, 2009). Therefore, only the ends of the fragments are sequenced. This results in a collection of read pairs that are separated by a known distance (the size of the original fragment) (Schatz et al., 2010). Paired-ends allow for assemblies larger than the length of the individual reads. This is accomplished by creating scaffolds – genome sequences reconstructed from contigs and gaps. Scaffolding allows for the joining of

two contigs even if there is no discernible overlap. Since the length is known between two paired-ends, the number of bases between two contigs that contain two paired-ends can be calculated. The scaffolding of two contigs with five sets of paired ends is given in Figure 21.

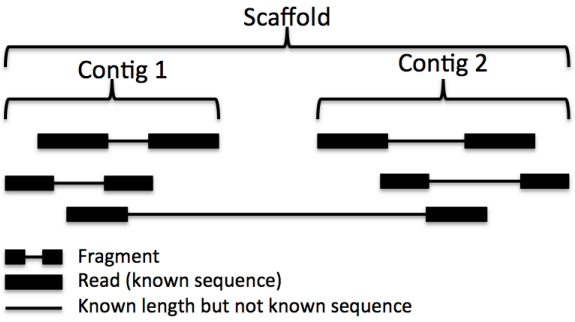

Figure 21 Example of scaffolding using two contigs and five paired-ends

Parallelization, however, would provide the best speed improvements by drastically decreasing the time needed to perform an assembly. In computing, parallelization is the act of performing many calculations simultaneously (Almasi & Gottlieb, 1989). The advent of multi-core processors and multi-processor computers has placed an increased emphasis on parallel computing. However, parallelizing a program is notoriously much more difficult to program than sequential ones (Patterson, 1998). New software bugs are introduced when one process or thread is trying to access a block of memory that is shared between all other processes. Also called race conditions, problems can arise when multiple processes try to read and write the same block of information at the same time. Resolving this situation

requires the use of locks. When one process is accessing a shared block of memory the memory location is *locked*. All other processes must wait until the memory is updated and *unlocked* before another process has access to that bit of information.

In this thesis, parallelization would prove to be most useful in the assembly method. The profiling tool *Callgrind* from the *Valgrind* suite was used to analyze the timing and call history of the program (Weidendorfer & Kowarschik, 2004). Based on the results, upward of 70% of the processing time is spent in the bookkeeping function (Appendix B.2). Using two threads to update the dictionary would greatly reduce the time necessary to traverse the `map`. One thread would start at the top of the dictionary and work down while another would start at the bottom of the dictionary and traverse up. Once they meet the dictionary is fully updated. This process is outlined in Figure 22.

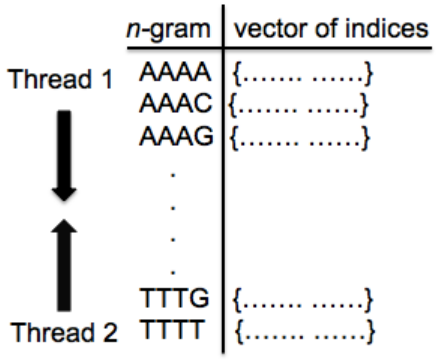

Figure 22 Overview of possible parallelization in the bookkeeping function

Bibliography

AMOS. (2010). Retrieved March 21, 2012, from amos.sourceforge.net/

Almasi, G. S., & Gottlieb, A. (1989). *Highly Parallel Computing*. Redwood City, CA: Benjamin-Cummings Publishers.

Baxevanis, A. D. (2004). *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins* (3rd ed., p. 560). Wiley-Interscience.

Cock, P. J. a, Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, *38*(6), 1767-71. doi:10.1093/nar/gkp1137

Cormen, T. H., Leiserson, C., Rivest, R., and Stein, C. (2009) *Introduction to Algorithms*. Cambridge (Massachusetts): MIT.

Davies, K. (2010). *The $1,000 Genome: The Revolution in DNA Sequencing and the New Era of Personalized Medicine* (p. 352). Free Press.

Ewing, B., & Green, P. (1998). Base-Calling of Automated Sequencer Traces Using Phred . II . Error Probabilities. *Genome Research*, 186-194. doi:10.1101/gr.8.3.186

Freeman, S. (2011). *Biological Science*. (S. Freeman, Ed.) (4th ed., p. 1320). San Francisco, CA: Benjamin Cummings.

Gerlinger, M., Rowan, A. J., Horswell, S., Larkin, J., Endesfelder, D., Gronroos, E., Martinez, P., et al. (2012). Intratumor Heterogeneity and Branched Evolution Revealed by Multiregion Sequencing. *New England Journal of Medicine*, *366*(10), 883-892. doi:10.1056/NEJMoa1113205

Horner, D. S., Pavesi, G., Castrignanò, T., De Meo, P. D., Liuni, S., Sammeth, M., Picardi, E., et al. (2010). Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Briefings in bioinformatics*, *11*(2), 181-97. doi:10.1093/bib/bbp046

Huang, X., & Madan, a. (1999). CAP3: A DNA sequence assembly program. *Genome research*, *9*(9), 868-77. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=310812&tool=pmcentrez&rendertype=abstract

Isilanes. (2007) *DNA Base Pair*. Digital image. Wikimedia Commons. <http://commons.wikimedia.org/wiki/File:GC_DNA_base_pair.svg>.

Kotsiantis, S. B., Kanellopoulos, D., & Pintelas, P. E. (2006). Data Preprocessing for Supervised Leaning. *Journal of Computer Science*, *1*(2), 111-117.

Lakdawalla, A. (2007) *Radioactive Fluorescent Sequence*. Digital image. Wikimedia Commons.<http://commons.wikimedia.org/wiki/File:Radioactive_Fluorescent_Seq.jpg>.

Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., et al. (2010). De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 265-272. doi:10.1101/gr.097261.109.20

Mardis, E. R. (2008). Next-generation DNA sequencing methods. *Annual review of genomics and human genetics*, *9*, 387-402. doi:10.1146/annurev.genom.9.081307.164359

Medvedev, P., Georgiou, K., Myers, G., & Brudno, M. (2007). Computability and Equivalence of Models for Sequence Assembly. *Algorithms in Bioinformatics*, *4645*, 50-64. Retrieved from http://www.citeulike.org/user/niallhaslam/article/1957796

Miller, J. R., Koren, S., & Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics*, *95*, 315-327. doi:10.1016/j.ygeno.2010.03.001

Mullikin, J. C., & Ning, Z. (2003). The Phusion Assembler. *Genome Research*, 81-90. doi:10.1101/gr.731003.

Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, *2*(2), 275-90. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/7497129

NCBI Reference Sequence (RefSeq). (2012). Retrieved March 25, 2012, from http://www.ncbi.nlm.nih.gov/RefSeq/

NCBI: Assembly Basics. (2012). Retrieved March 27, 2012, from http://www.ncbi.nlm.nih.gov/projects/genome/assembly/assembly.shtml

Narzisi, G., & Mishra, B. (2011). Comparing de novo genome assembly: the long and short of it. *PloS one*, *6*(4), e19175. doi:10.1371/journal.pone.0019175

Patterson, D. (1998). *Computer Organization and Design* (Second Ed., p. 715). Morgan Kaufmann Publishers.

Pevzner, P. a, Tang, H., & Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, *98*(17), 9748-53. doi:10.1073/pnas.171285098

Pop, M., Salzberg, S. L., & Shumway, M. (2002). Genome sequence assembly: algorithms and issues. *Computer*, *35*(7), 47-54. doi:10.1109/MC.2002.1016901

Pop, Mihai. (2009). Genome assembly reborn: recent computational challenges. *Briefings in bioinformatics*, *10*(4), 354-66. doi:10.1093/bib/bbp026

Pruitt, K. D., Tatusova, T., Klimke, W., & Maglott, D. R. (2009). NCBI Reference Sequences: current status, policy and new initiatives. *Nucleic acids research*, *37*(Database issue), D32-6. doi:10.1093/nar/gkn721

Schatz, M. C., Delcher, A. L., & Salzberg, S. L. (2010). Assembly of large genomes using second-generation sequencing. *Genome research*, 1165-1173. doi:10.1101/gr.101360.109

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J. M., & Birol, I. (2009). ABySS: a parallel assembler for short read sequence data. *Genome research*, *19*(6), 1117-23. doi:10.1101/gr.089532.108

Sutton, G. G., White, O., Adams, M. D., & Kerlavage, A. R. (1995). TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, *1*(1), 9–19. doi:10.1089/gst.1995.1.9

Weidendorfer, J., & Kowarschik, M. (2004). A tool suite for simulation based analysis of memory access behavior. *Proceedings of the 4th International Conference on Computational Science*. Retrieved from http://www.springerlink.com/index/N6C3ENA60K28LANX.pdf

Zerbino, D. R., & Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, *18*(5), 821-9. doi:10.1101/gr.074492.107

Appendix A

Functionality

Below is an explanation on the functionality and use of the program. The running of the program assumes that the program name is `ngassemble`. An explanation of the configuration file is outlined in Appendix A.2.

A.1 How to Run the Program

```
Usage: ngassemble [CONFIG] [OVERRIDES]

CONFIG is the name of the configuration file.

OVERRIDES follows the format -O[KEY]=[VALUE] where KEY is the key name
      in the configuration file and VALUE is the value to change the
      key to. This overrides the value in the configuration file.
      OVERRIDES is useful when running the program in a script and
      changing the value of a certain key each iteration is unwieldy.
      Multiple overrides (-O) can be used to change multiple keys.
```

A.2 Configuration File

ngramSize is the size of the *n*-gram to analyze. The ngramSize is also used in preprocessing. If the read length is less than the ngramSize the read is discarded.

dataFile is the data file to analyze. This can either be a FASTQ or FASTA file. FASTQ files will be assembled. FASTA files are used for analysis of *n*-grams.

machineSequencer is the machine type used to generate the data file. This can be either sanger, 454, solexa, or illumina. Each machine type has a different QV calculation.

percentN is the maximum percentage of N allowed in data preprocessing. The value is represented as a percent. For example, the value 76 is interpreted as 76%. A read is discarded if percentN < the actual percentage of N in the read. Therefore, a percentN of 100 allows for no preprocessing of the N percentage.

qualityScore is the minimum average quality score to allow in data preprocessing. If the average read QV is less than the qualityScore the read is discarded. A qualityScore of 0 allows for no preprocessing of the QV.

percentMiss is the maximum percentage of mismatches allowed when attempting to join two reads. The value is represented as a percent. For example, the value 5 is interpreted as 5%. If the percentage of mismatches is greater than the percentMiss the join will not be made.

runPerform is a print option to only print out the performance metrics. This includes the number of reads excluded based on *n*-gram size, QV score, and percentage of Ns. The final longest contig and N50 value is also printed after completion of the assembly. This option is useful when running in bash scripts since it prints out the minimal amount of information.

runPrint is a print option to only print out the map after completion of the assembly. The excluded reads are also printed.

runPercentN is a print option to print out the percentage of Ns for each read in the read list. The excluded reads are also printed.

runQV is a print option to print out the average QV score for each read in the read list. The excluded reads are also printed.

runSequence is a print option to print out all the sequences and their QV score for each read in the read list. The excluded reads are also printed.

runGeneric is a print option to print out the runPerform in addition to the time it takes to complete the assembly.

printConfigDB is a print option to print out the configuration file. The excluded reads are also printed.

printMap is a print option to print out the *n*-gram map. The excluded reads are also printed.

printFrags is a print option to print out the read list. The excluded reads are also printed.

printContigs is a print option to print out the contigs after assembly. This is useful when the genome is not fully assembled and the contigs need to be imported into another program for coverage analysis. The excluded reads are also printed.

printAnalyses is a print option to print out the final read list and map. The excluded reads are also printed.

printLongest is a print option to print out the longest contig after assembly. The excluded reads are also printed.

printN50 is a print option to print out the N50 score after assembly. The excluded reads are also printed.

printAvgQV is a print option to print out the average QV score for each read in the read list. The excluded reads are also printed.

outputContigs is an option to export the contigs to a separate file for additional analysis. This is useful when the genome is not fully assembled and the contigs need to be imported into another program for coverage analysis. The printContigs flag must also be set to TRUE. The excluded reads are also printed.

Appendix B

UML and Profiling

In Computer Science, UML, short for Unified Modeling Language, is a way to show relations and structures between different classes in a program. A UML diagram lists the classes and member data used in a project. The UML diagram for this thesis was constructed using ArgoUML (Medvedev et al., 2007; E. W. Myers, 1995; Mihai Pop, 2009).

B.1 UML

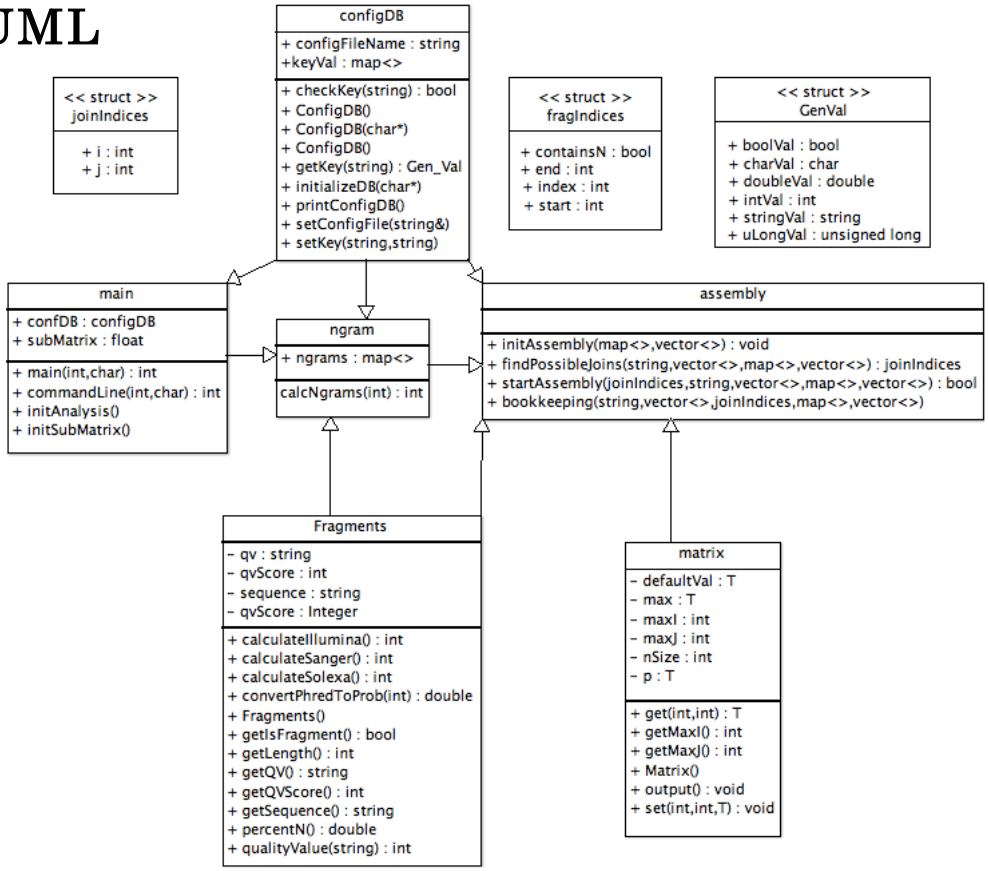

Figure 23 UML diagram for project

B.2 Callgrind Profiling Analysis

Profiling was conducted using *Callgrind*, part of the *Valgrind* suite of utilities (Weidendorfer & Kowarschik, 2004). Timing and call history analysis was recorded and a graph was generated. The figure is displayed below.

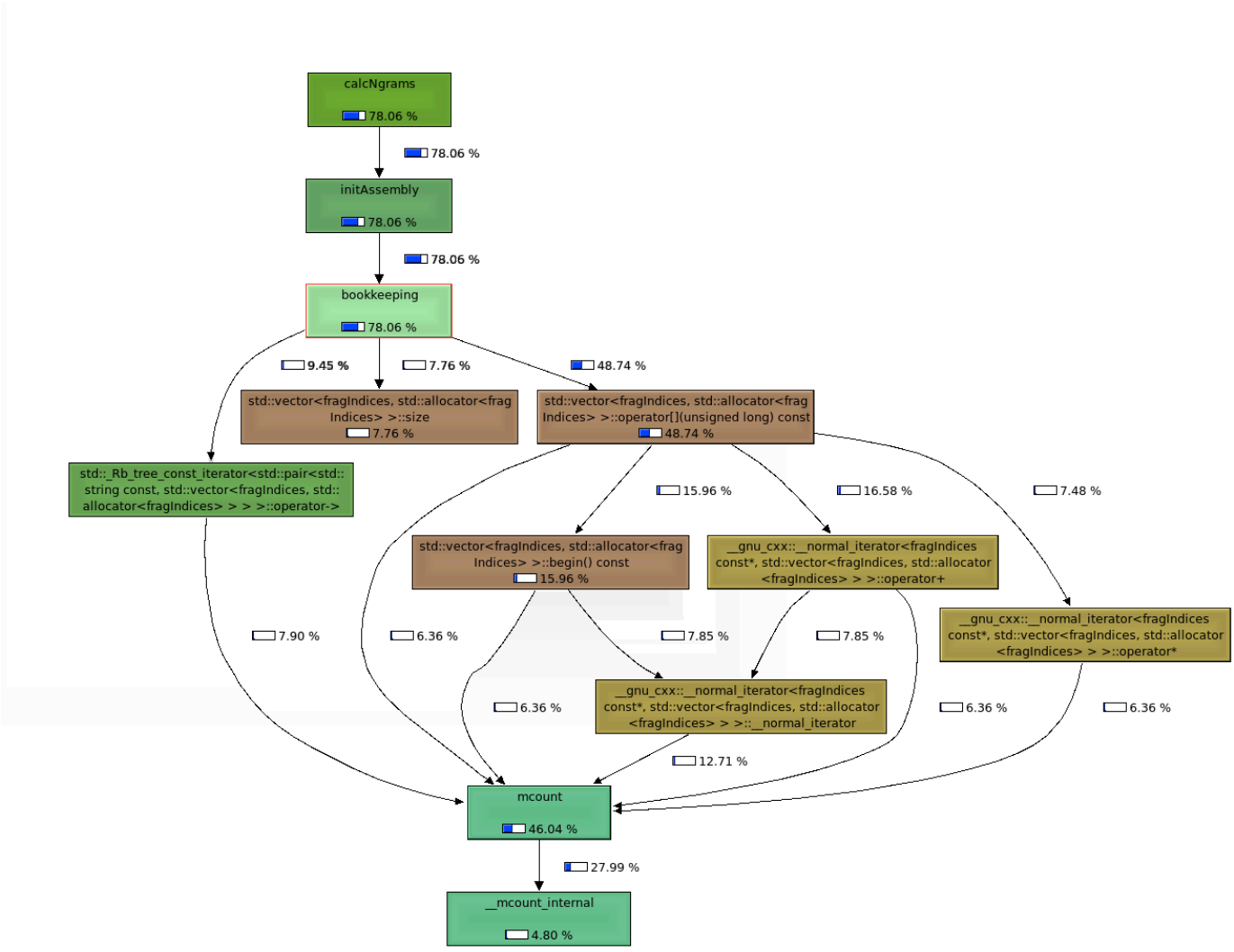

Figure 24 *Callgrind* analysis of the assembly method

Appendix C

Glossary

Alignment A way of rearranging and comparing DNA to identify similarities between the sequences.

Base pairing In DNA, when a nucleotide on one strand interacts with an appropriate base on the other strand.

Bioinformatics The interdisciplinary field between biology, statistics, and computer science.

Contig A set of overlapping DNA segments that together represent a consensus region of DNA.

Dideoxynucleotides Modified deoxynucleotides used in the Sanger method of DNA sequencing to terminate the strand. This prevents the addition of further nucleotides.

FASTA The *de facto* standard in bioninformatics for storing DNA and protein sequence data.

Flow cell A liquid stream which carries and aligns the DNA. This aids in binding the DNA to enzymes and the cell surface for manipulation.

Fragment Short sections of DNA whose sequence is unknown.

Map A C++ Standard Library associative container. A map stores information by a combination of a *key* and corresponding *maped value*.

N50 A metric used to evaluate an assembly. It is defined as the largest number L such that the combined length of all contigs of length greater than or equal to L is at least 50% of the total length of all contigs.

NCBI The National Center for Biotechnology Information. The NCBI stores genome sequencing data from thousands of organisms.

Parallelization A form of computation where many calculations are performed simultaneously.

PCR Polymerase chain reaction is a scientific technique in molecular biology to amplify a single or a few copies of a piece of DNA across several orders of magnitude, generating thousands to millions of copies of a particular DNA sequence.

Process In parallel computing, an instance of a computer program that is being executed. A process can be divided into multiple threads.

Purines The nucleotides adenine (A) and guanine (G).

Pyrimidines The nucleotides cytosine (C) and thymine (T).

Race conditions In parallel computing, when multiple processes try to access shared blocks of memory. Problems can arise when multiple processes try to read and write the same block of information at the same time.

Reads Short fragments of DNA outputted from genome sequencers whose sequence is known.

Sequence assembly A concentration of bioninformatics that refers to aligning and merging reads of a much longer DNA sequence in order to reconstruct the original sequence.

Scaffold In whole-genome sequencing , a portion of DNA reconstructed from contigs and the appropriate gaps.

Thread In parallel computing, the smallest subunit of processing allowed by the operating system. A process can be divided into multiple threads.