SCALABLE MAPPING-FREE ALGORITHMS AND K-MER DATA STRUCTURES FOR GENOMIC ANALYSIS

A Thesis
Presented to
The Academic Faculty

by

Peter A. Audano III

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Biology

Georgia Institute of Technology
August 2016

SCALABLE MAPPING-FREE ALGORITHMS AND K-MER DATA STRUCTURES FOR GENOMIC ANALYSIS

To my wife,

Dawn S. Audano,

for her patience and sacrifice in my pursuit of academic achievement.

ACKNOWLEDGEMENTS

I want to thank my committee members for taking time from their busy schedules to advise me through this process.

Dr. Irving King Jordan is one of the most critical professors I have ever known. His talent for identifying the weakness of an approach was one of the most important tests my work had to pass. His class, *Computational Genomics*, is one of the most feared among bioinformatics graduate students as it teaches students to take on big problems, build solutions, and to do it all with minimal supervision. I knew that Dr. Jordan would not let me pass easily through this program without making solid contributions to the science of bioinformatics, and so it was important for me to have him on my committee.

Dr. Brian Hammer brings a biological perspective to my committee. Much of my work has focused on bacterial pathogens, and his *Prokaryotic Molecular Genetics* class provided me with a critical understanding of their genetics and the significance of what I was doing.

Dr. Srinivas Aluru has an engineering background, and his experience applying it to science was another perspective that I needed in my committee meetings. He has also been the principal investigator for a project using k-mers to correct sequence read errors, and this familiarity with k-mer algorithms is rare.

Dr. Gregory Gibson made most of this research possible by administering the NIH T32 training grant that funded my work. As a result, I was able to focus on the problems that I thought were interesting, and I was able to make significant advancements with that freedom. Dr. Gibson is also an accomplished geneticist, and his feedback on my work has been insightful.

Dr. Fredrik Vannberg is one of the most forward-thinking professors I have ever met. His eagerness to develop new techniques was an inspiration from the first day I went to work in his lab. I pursued k-mers in the early days under his direction. KAnalyze was born from that effort, and everything else was made possible because if it.

Many students passed through the Vannberg lab while I was there, and they brought fresh perspectives and ideas. In the early days, Vivek Sagar Krishnamurthy Radhakrishna and Benjamin Metcalf were an inspiring pair to work with. Benjamin Metcalf left to pursue a career at the Centers for Disease Control and Prevention (CDC), and he played a key role developing a convincing test case for my work on *Streptococcus pneumoniae*. Sharing ideas with them is still the highlight of my graduate experience, and I can only hope to work with scientists who are so brilliant and so motivated.

As an undergraduate studying computer science, Dr. Paula Checchi helped reinforce a love and fascination for biology through her introductory biology course. Before her class, I did not know that the field of bioinformatics existed, and by the end of it, I was planning my foray into it. She also developed a special topics bioinformatics class, and it was my first course in the subject. Computer science instructors Dr. Bob Harbort and Dr. Briana Morrison were also very influential at that time.

The influence of grade school teachers cannot be underestimated. Special thanks goes to Mr. Stephenson, Mrs. Alice Bolstad, Mrs. Swanson, Mr. Charlie Thomas, and Mrs. Mara Uman Hixon. These people played an influential and positive role in one of the most tumultuous times in my life.

I would also like to thank high school teachers Mr. Scott Flamand and Mrs. Nancy Maple for my first classes in Biology and computer programming. Because of their thoughtful instruction, I learned that I had a fascination for these subjects, and over the years, that fascination has not waned.

I cannot go without thanking my parents and my sister, who were supportive of me through this endeavor. Special thanks also goes to my father-in-law, Mr. Randy Suchke, who has been supportive of our family while I earned a student's stipend.

The sincerest thanks goes to my wife, who a endured marriage with a graduate student. She has been patient through my long work days and work weekends while managing the home and raising our daughter, Laurel. This was as much of a sacrifice for her as it was for me, and she has earned this degree as much as I have. She also contributed the logos for several of my software applications, and she created the graphics for Figure 14.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

ANI	Average nucleotide identity.
API	Application programming interface.
bp	Base-pair.
canonical k-mer	A single k-mer representation of the k-mer and its reverse complement.
CDC	The Centers for Disease Control and Prevention.
CRISPR	Clustered regularly interspaced short palindromic repeat.
Gb	Gigabase; one billion nucleotide bases.
GB	Gigabyte; 1024^3 bytes.
HGT	Horizontal gene transfer.
IKC	Indexed k-mer count.
indel	An insertion or deletion in a nucleotide sequence.
JVM	Java virtual machine.
Kb	Kilobase; one thousand nucleotide bases.
LPA	Line probe assay.
Mb	Megabase; one million nucleotide bases.
MDR	Multi-drug resistant.
MLST	Multilocus sequence typing.
MRSA	Methicillin-resistant Staphylococcus aureus.
NGS	Next-generation sequencing.
NHGRI	National Human Genome Research Institute.
SNP	Single nucleotide polymorphism.
TB	Tuberculosis.
WGS	Whole genome sequencing.
WHO	World Health Organization.
XDR	Extensively drug resistant.

SUMMARY

An organism's DNA sequence is a virtual cornucopia of information, and sequencing technology is the key to unlocking it. The past few decades have been witness to meteoric growth of these technologies and the volume of data they produce. New industries are forming around it, existing ones are changing as a result of it, and modern medicine is on the precipice of a genomic revolution.

Turning this deluge of data into useful information is already a challenging task with advances in sequencing technology far outstripping advances in semiconductors, and this trend shows no signs of stopping. Incremental advances are being made in sequence analysis techniques, but progress is far too slow to keep up with the volume of data delivered by modern sequencing platforms. This gap is often filled by allocating more computing resources in the form of distributed computing platforms, and this can quickly become prohibitive. Because medicine requires a quick answer and science often has limited funding, the analysis bottleneck is a major concern.

Instead of finding new ways to dedicate more computing resources to the problem, I am interested in streamlining the process. In this dissertation, I explore methods to make the analysis faster and more efficient. My ultimate goal is to create algorithms that can run on a standard computer workstation, and when necessary, make the most of expensive distributed and cloud computing resources.

Many analysis pipelines start by aligning sequence reads to a reference or assembling them into long consensus sequences, but it can take several hours to analyze a single sample on a workstation computer. Instead aligning or assembling sequence reads, the approaches described in this dissertation transform and analyze the sequence data without alignments. These *alignment-free* approaches often improve

performance by an order of magnitude or more.

The first step for many alignment-free approaches involves transforming the sequence reads into k-mers, which are short overlapping fragments of uniform size. If the size, k, is 48, then all substrings of length 48 are extracted from each sequence read and counted. The resulting frequency of each k-mer can then be used as evidence for genomic analysis techniques.

When I started as graduate student, I began by working on a program to transform sequence data to k-mer frequencies. The resulting software, KAnalyze, was incredibly flexible because of its software architecture and approach to solving the k-mer counting problem. This became the foundation for the rest of my graduate work because it was possible to test new ideas that required data structures and transformations not commonly used today.

After my initial work with k-mers, I was connected with the Centers for Disease Control and Prevention (CDC) *Mycobacterium tuberculosis* (*M. tuberculosis*) science team. They were interested in replacing existing software with an alignment-free approach based on k-mers. The new software significantly reduced analysis time, and it reduced errors. K-mers are more rigid than sequence aligments, so I had to find a way to correct for mutations in the samples. This resulted in a novel algorithm that could identify *single nucleotide polymorphism* (SNP) and *insertion/deletion* (indel) mutations, and it still took far less time than the alignment approach.

I am not aware of an error correction algorithm with k-mers that does not employ a simple hamming distance calculation, and therefore, is capable of handling indel variants. The CDC *M. tuberculosis* project solved this problem in a naïve way; when a variant was detected, it took 4 paths and assumed that it might be a SNP, insertion, deletion, or no variant. The result is an algorithm that runs with a computational time complexity of $\mathcal{O}(4^n)$. Although this worked well on the short reference sequences analyzed for this project, it would never scale to larger references.

After some thought and experimentation, I realized that the reconstruction algorithm I had created was generating a sequence that was ostensibly related to the reference, and comparing two sequences to determine how well they match is the fundamental task of alignments. Instead of taking multiple paths for each base mismatch, could it be possible to guide the rebuilding algorithm by aligning the dynamically constructed sequence with the reference? Although it is not 100% alignment-free, this approach would significantly reduce the computational burden of modern methods, which aligns each of the sequence reads to the reference. What was more exciting is that it might be able to pick out variants blind to sequence read alignments, such as dense SNP loci or large insertions. This work culminated in Kestrel, which is a novel first-in-class variant caller application that uses this idea.

The chapters that follow tell this story from the current state of the art to novel applications of this new technology that are under development today.

CHAPTER I

AN INTRODUCTION TO SEQUENCE ANALYSIS

1.1 Abstract

The arrangement of letters in our DNA has a profound impact on our lives, and in some ways, has coded our own fate from zygote to senior citizen. It determines our appearance, what diseases we are likely to suffer from, and it even informs our personalities. The arrangement of DNA letters in the organisms around us has an equally significant impact. The code of pathogens gives them tools for infecting hosts, evading immune systems, and mitigating the effects of antibiotics. Manipulating the code has led to a biotech revolution that has changed the way we farm and eat. It is difficult to understate the impact that the sequence of these letters has on human health and all living things.

Sequencing equipment is becoming smaller, more affordable, and capable of multiplexing more data per run, but the sequence reads generated by modern technology are relatively tiny compared to the length of the molecule it came from. The human genome consists of 3 billion bases, and with sequence read lengths in the hundreds, finding useful information in a torrent of short reads is no small task. It requires clever approaches, and often, expensive computing resources such as distributed and cloud computing systems.

Modern analysis methods often align sequence reads to a reference. Since it requires millions of inexact string comparisons, generating an alignment can require many CPU-hours to complete. However, analyzing the resulting structure is often far more efficient. Fortunately, there is emerging class of *alignment-free* algorithms that offers alternatives that can decisively outperform methods based alignments.

1.2 *Next-generation sequencing*

Sanger sequencing was originally described in 1975 [105] as a sequencing method using gel electrophoresis and radioactively labeled di-deoxy nucleoside bases. It was later improved by using one dye for each base, and so the whole reaction could be done and electrophoresed together [110]. This had the added advantage of making it easier to analyze the result with a computer since merging 4 lanes was no longer required. Gels were replaced with capillaries, and at its height, 96 strands of DNA could be sequenced simultaneously by a single machine.

Sanger sequencing was the technology that made the Human Genome Project a success [81]. Subsequent efforts to characterize common variation within the human genome, known an the HapMap project [22], also relied heavily on this technology. These early efforts continue to have an unmeasureable impact on biological sciences, but expanded efforts, such as the 1000 Genomes Project [21], required far more sequencing data than Sanger sequencing offered.

In 2005, the first of a new wave of sequencing technologies was published: pyrosequencing [82]. This approach is implemented by the Roche 454 and the Thermo Fisher Scientific Ion Torrent sequencers. Soon after pyrosequencing, reversible terminator technology was released [7], and this is used today by the Illumina sequencing platforms, which can produce up to 1800 Gb in a single run [50]. These and other approaches developed since 2005 are collectively known as *next generation sequencing* (NGS) technologies.

Because it runs many reactions in parallel, NGS sequencing has higher throughput per run, and therefore, produces larger quantities of data in a shorter amount of time. This increased throughput, and therefore depth of sequence coverage, has enabled many new types of studies that were not possible before [81, 87].

The higher throughput and lower cost of NGS has also led to an explosion in the demand for sequencing. Once relegated to well-funded laboratories, sequencing

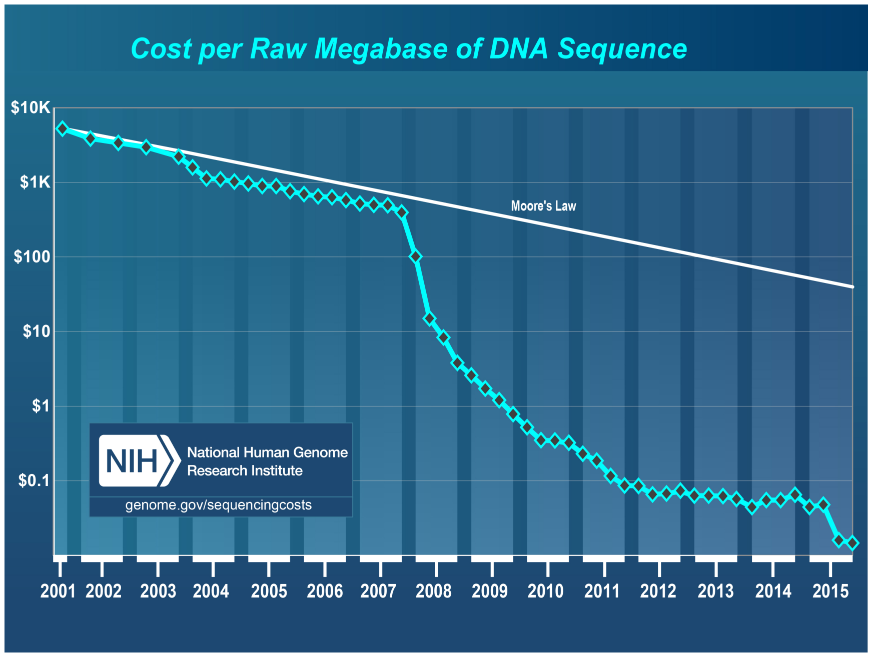

Figure 1: Cost to generate 1 Mb of sequencing data compared to Moore's law.

experiments have become more accessible and even standard practice. More affordable and impressively capable benchtop sequencers are now available so that smaller laboratories can run their own projects [77]. For example, these sequencers could be used to fully characterize the causative agent of an infection in less than 24 hours [48].

There is a monumental effort underway to find ways to make sense of this data and to answer biological questions once out of reach. For example, the Centers for Disease Control and Prevention (CDC) has earmarked $150 million to improve bioinformatic and NGS approaches for molecular epidemiology [41].

As sequencing throughput increases, so does the the demand for sequence analysis. Each year, National Human Genome Research Institute (NHGRI) publishes a well known graphic depicting the cost of sequencing against moore's law [89], which estimates the rate of semiconductor advancement. Figure 1 shows this graphic as of 2015. Because of this explosion of data availability, it now takes many computer processors to analyze that data in a reasonable amount of time, and many biological studies must rely on cloud computing resources to do it [28].

With improving NGS technologies and the emerging trend for benchtop sequencers, the amount of sequencing data that will be generated in coming years will only grow. Since semiconductors cannot keep up with this pace, we must turn to algorithms that make the most of the processing power and memory we do have. Without it, the cost of analysis will outstrip the cost of sequencing, and this is a serious impediment to the progress of biotechnology.

1.3 Sequence alignment

The sequence reads produced by NGS technologies do not represent a full strand of the original DNA; they are tiny fragments of it. For a strand of DNA that may contain many megabases, each sequence read typically covers 100 to 500 bases of it. Any one of these sequence reads reveals very little about the sample, so they must be aggregated in some meaningful way.

To put the sequence data back into a useful context, the reads are typically aligned with a reference genome. Inference about the sample is then made on the aligned reads using tools such as the GATK [85] HaplotypeCaller and FreeBayes [40]. The majority of variant calling pipelines follow this pattern [96, 92]. Figure 2 shows an alignment with highlighted variants.

Although the alignment approach is the *de facto* standard, it does have some significant disadvantages. Because it is essentially a string-matching problem, finding the location of sequence reads is a complex and CPU-intensive process. To simplify this problem, the BWA [68, 69] and SOAP2 [74] aligners use burrows-wheeler transforms to generate a compressed index of the reference, and the Bowtie [64, 63] aligner uses seed sub-sequences. These are among the most well respected tools, but the alignment process can still take many CPU hours to complete.

No matter how advanced the alignment algorithm is, it can never correctly assign

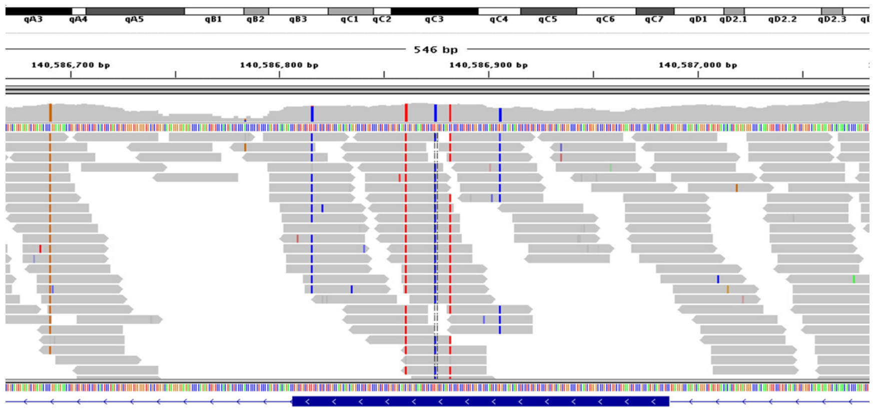

Figure 2: Depiction of a sequence alignment. The gray bars are sequences aligned to a reference. Blue and red lines on the sequence reads denote loci where the read and the reference are different. This image is a screenshot of The Broad Institute's Integrative Genomics Viewer (IGV).

sequence reads when it differs too much from the reference without incorporating some assembly approach. This is especially true for sequence reads in a large insertion because they will have no bases in common with the reference.

1.4 Alignment-free analysis

There is an alternative class of alignment-free algorithms where sequence reads are not directly mapped to a reference. Typically, these approaches transform sequence data to k-mers, which are short overlapping fragments of uniform length. Each unique k-mer in a set of sequence reads is then counted, and the map of unique k-mers to their frequency serves as the basis for genomic analysis. CHAPTER II will discuss k-mer counting in greater detail.

Alignment-free approaches benefit by skipping the alignment step. Furthermore, k-mers can be treated numerically, and so expensive string-matching operations are not needed. Where an alignment and an alignment-free algorithm produce similar results, the alignment-free approach tends to be much faster.

By avoiding alignments, many CPU cycles can be reclaimed allowing for deeper and more frequent processing of data. As sequencing machines produce more data every year, these approaches represent a promising way forward. This is even more critical with the gaining popularity of benchtop sequencers. Not every scientific or medical lab can afford to offload analysis to a cloud, and even in CLIA certified platforms, there is still and issue of data privacy.

1.5 Chapters

CHAPTER II discusses how sequence data is transformed into k-mers, and it outlines key data structures and algorithms. A k-mer toolkit is presented, KAnalyze [4], and this is the software that enable the analysis methods described in subsequent chapters.

CHAPTER III outlines an alignment-free bacterial typing project that uses k-mers and KAnalyze. This project created a novel, although inefficient, sequence correction algorithm. Solving this efficiency problem led to a more general method of variant calling, which is described in CHAPTER IV.

CHAPTER IV presents a novel variant calling algorithm that does not require sequence alignments, de Bruijn graphs, or *de novo* assembled sequences. A software implementation, Kestrel, characterizes variant in regions where alignments fail.

CHAPTER V discusses novel applications k-mer technology, and it shows how Kestrel can be used in a wide variety of applications. Future applications of this technology are being developed, and they are discussed here.

CHAPTER VI concludes this dissertation and summarizes my contributions to sequence analysis.

CHAPTER II

K-MERS AND KANALYZE

2.1 Abstract

A k-mer is a sequence that is exactly k bases long, and a larger sequence can be transformed to a set of k-mers by finding all possible substrings of length k. Viewing sequence data as these shorter constructs is one of the keys to efficient alignment-free genomic analysis. The transformation to k-mers takes less time than aligning reads, and it has the power to replace alignments for an emerging class of algorithms. These algorithms are further streamlined by avoiding string operations because k-mers have a native binary representation.

Often, the k-mer frequency of a data set is used in alignment-free algorithms, and this requires matching and counting all instances of each unique k-mer. Given millions of sequence reads and many gigabytes of data, counting becomes a hopelessly inefficient task if done naïvely. Clever algorithms are needed to make counting a tractable problem, and this is especially true when the set of k-mers cannot not fit into the memory of one machine.

KAnalyze [4] is the result of a software development effort to create not only a k-mer counter, but also a toolkit of reusable components. This software is not the first in its class, but it is by far the most flexible, and this flexibility has made it an indispensable tool for the development of novel analysis algorithms.

2.2 Introduction

To study the genomics of some biological sample, DNA is extracted and sequenced. There are several steps involved, but the result is typically one or more FASTQ files with millions of short sequence reads and quality scores for each base-call. Ideally, these sequences cover the genome several times over and are a good representation of the DNA in the original sample.

These sequence reads must be transformed in some meaningful way in order to extract useful information about the genome the sample represents. Many approaches start by mapping these reads to a reference sequence, as outlined in Section 1.3. However, this chapter is concerned with a different kind of transformation: k-mers.

Interesting characteristics of a genome can be revealed by examining k-mer frequencies, such as the genome size and read coverage [76]. Chor *et al.* [19] show that the 11-mer frequency distribution of the human genome has a distinct tri-modal distribution, which suggests that certain sequence characteristics play specific roles. This is especially true for the CpG motif, which tends to occur only in CpG islands, as demonstrated by the same study with 8-mers. More recently, methods based on k-mers have led to significant advances in metagenomics [118], RNASeq analysis [98, 12], bacterial phylogenetics [39, 38].

The algorithms presented in this paper are mostly concerned with k-mer frequencies, which are represented as a set of unique k-mers in the sequence data and the number of times each was found. This set is another view on the sequence data, and alignment-free algorithms can make inferences based in it.

The remainder of this chapter will discuss how k-mers are generated and handled. Subsequent chapters will outline various analysis methods using k-mer data.

2.2.1 Generating k-mers from sequences

For illustration purposes, the k-mer size in the following examples will be 4. In real sequencing data, this length is typically in the range of 30 to 80. Section 2.2.3 will discuss choosing an appropriate k-mer size.

K-mers can be extracted from a sequence by reading it from left to right and taking each k bases. The first k-mer starts on the first base, the second k-mer starts on the second base, and so on until the last base of the sequence is added to a k-mer. This is essentially a sliding-window of size k over the sequence. Figure 3 shows an example of k-mers generated from a sequence.

Example: k = 4

```
GCGTTGCGTTA
GCGT
 CGTT
  GTTG
   TTGC
    TGCG
     GCGT
      CGTT
       GTTA
```

Figure 3: K-mer generation by a sliding window.

2.2.2 Canonical k-mers

K-mers represent one strand of DNA, but sequencing protocols typically sample from both strands. Therefore, each region of k bases is represented by two complementary k-mers. For example, when matching the frequency to k-mers in a known gene sequence, only half of the sequence reads in that gene are reflected in the those k-mers. To get the true k-mer depth, both the k-mer and its reverse complement need to be considered.

One approach is to transform each k-mer to its canonical form, which is defined as the lesser of a k-mer and its reverse-complement where "lesser" is defined by sort order [60]. For example, the canonical representation of "CAAT" is "ATTG" (its reverse-complement). The canoncial representation of "ATTG" is itself since its reverse complement is lexically greater.

When these *canonical k-mers* counted, both DNA strands are represented without duplicating data. This tranformation has the added benefit of reducing the original k-mer frequency set to approximately half its size. However, useful strandedness information is lost, and so it may not be suitable for some algorithms.

2.2.3 Choosing the size

Choosing the k-mer size has been heavily researched for *de novo* genome assembly. De Bruijn graphs are constructed by letting the k-mers of each sequence be a vertex and letting edges connect the vertices where the k-mers are adjacent in sequence reads. Repeat structures and homologous regions create loops in the graph, and so the longest possible k-mer should be used to span them. Some software applications, such as KmerGenie [18] and an optimizer program for Velvet [120] attempt to pick the optimim k-mer size given the data.

In the absence of read errors, the k-mer size might be the size of sequence reads. However, since read errors do occur, the optimal length depends on the error rate, read lengths, and the length of k-mers required to resolve repetitive sequence [18, 120]. If larger k-mers must be used and the error rate causes a significant loss of data, sequence reads could be corrected before they are transformed to k-mers [119]. This correction would increase the execution time, but it may still be faster than alignments.

Smaller k-mers have been found to be usful for other purposes. 16-mers may be adequate for bacterial species identification [48], and a k-mer size of 10 to 16 is useful for correcting sequence read errors [119].

2.2.4 K-mer Space

The analytic power of these approaches is related to the size of k-mer space, which is defined as all possible k-mers of size k. For example, there are 4^{31} unique k-mers in 31-mer space. If the human genome was a random sequence of bases, then the chance of randomly picking a k-mer that was found in it would be $3 \times 10^9 \cdot 1/4^{31}$, or

6.5×10^{-5}. Real genomes have structure that makes them non-random, however, an adequately chosen k and a good algorithm can often identify useful patterns in the data.

It is worth noting that this space inherently contains mutual information because a k-mer and its reverse complement are linked. However, by using the canonical representation (Section 2.2.2), this mutual information is removed, and the k-mer space is reduced to almost $\frac{1}{2}$ its original size ("almost" because some k-mers are their own reverse-complement).

When necessary, I will refer to the space of k-mers with size $k > 0$ as $\mathbb{K}_k$. This space is a set with size (cardinality) $|\mathbb{K}_k| = 4^k$. When k is 1, then this space becomes a set of the 4 bases. Technically, this space could have $k = 0$, in which case it is an empty set $\mathbb{K}_k = \emptyset$, however this is not useful. I restrict k to values greater than 0 for this discussion.

2.2.5 Advantages and disadvantages

One key advantage of k-mers is that they naturally represent fragments of the genome in a compressed form. As will be shown in Section 2.3.1, each k-mer can be represented as a number. Algorithms and data structures are far more efficient on these numbers than they are on their string equivalents. This numeric structure is made possible by a binary representation that can also be transformed with the bitwise operations available in many programming languages.

Many bioinformatic algorithms align reads to a reference. This takes considerably more time and more computing resources to complete than transforming the data to a set of k-mer frequencies because of the string operations that must be performed to match the read and the reference. With efficient data structures, such as the one outlined in Section 2.3.6, these frequencies can be rapidly queried.

K-mers are a rigid structure. When any base is changed, such as sequence read errors or genomic variants, the k-mers that touch that base are different. It is difficult to perform fuzzy matching with k-mers, and when it is done, it typically comes in the form of a limited hamming distance. If more than one variant occurs in the region of a k-mer, such as 3 or 4 SNPs, or if the sequence contains indels, attempting to apply this type of approach with k-mers becomes unwieldy and inefficient. Generally, a k-mer matches a region of k bases, or it does not, and algorithms have to work around it. It may appear that k-mers are unsuited for nothing but the most trivial analysis tasks, however, the algorithm presented in CHAPTER IV shows how k-mers can characterize genomic events that cause even the best alignment algorithms to fail.

The most significant disadvantage is that some valuable information is lost when sequences are shredded to k-mers. This is especially true when the sequence reads are used to generate k-mer frequencies. In this case, each k-mer is a snapshot of k bases from a sequence read, and it is irrecoverably separated from the rest of the data in that read. Furthermore, paired-end context is lost. De Bruijn graphs preserve some linkage among k-mers of a read, and SPAdes [5] preserves paired-end information by using paired de Bruijn graphs. In most cases, however, this causes a significant problem when dealing with genomic regions that share homology with other regions, such as paralogues or repeats.

In the future, hybrid approaches may be able to use k-mers to increase the speed of analysis, but preserve sequence read information to improve it. Very little has been done with such approaches, but I think some of the best algorithms in the future will operate this way.

2.3 Methods and algorithms

The algorithms and data structures described in this section are implemented by KAnalyze [4], a k-mer toolkit that I developed.

2.3.1 Binary and numerical representation

K-mers are not strings: they are numbers. For each k-mer, there is an integer $V \in \mathbb{Z}, 0 \leq V < 4^k$. This relationship is bijective, so the value V maps back to the original k-mer string. This mapping is unambiguous as long as k is known.

Just as the decimal system is base-10 with 10 symbols, [0-9], there is a base-4 system with symbols [ACGT]. Let A = 0, C = 1, G = 2, and T = 3. By assigning U = T = 3, RNA sequences may also be represented. For each letter, its value is multiplied by 4 raised to a power for the position of the letter. This is done for all bases in the k-mer, and the values are summed to obtain V. See Equation 1 for the transformation and Figure 4 for an example.

$$
\begin{aligned}
V &= \sum_{i=1}^{N} v(s_i) \cdot 4^{N-i} && (1) \\
V &:= \text{Numeric value of a k-mer} \\
s &:= \text{Character string representation of a k-mer} \\
N &:= \text{Number of characters in } s \\
s_i &:= \text{Character in } s \text{ such that } s_1 \text{ is the left-most character and } i \in 1 \ldots N \\
v(s_i) &:= \text{Assigns a value to a character, } s_i \ (A = 0, C = 1, G = 2, T = 3, U = 3)
\end{aligned}
$$

More formally, let $\mathbb{K}_k$ be the set of all possible k-mers of size k (the k-mer space) and K_k be one k-mer $K_k \in \mathbb{K}_k$. For each possible k-mer size $k > 0$, there exists a bijection between a subset of integers $\mathbb{Z}$ and $\mathbb{K}_k$, such that $\forall K_k \in \mathbb{K}_k \exists!\ V \in \mathbb{Z},\ 0 \leq V < 4^k$.

This numeric representation is more compact, and each k-mer takes exactly 2 binary bits because there are 4 bases ($\log_2 4 = 2$). Since each byte is 8 bits, 4 bases can be compacted into a byte. The string representation consumes one byte per base, so the numeric representation requires $\frac{1}{4}$th of the space with no loss of data.

Example: GTTA

$$\begin{array}{rcllll} \text{i} & = & 1 & 2 & 3 & 4 \\ \text{s} & = & \text{G} & \text{T} & \text{T} & \text{A} \\ \text{v}(s_i) & = & 2 & 3 & 3 & 0 \end{array}$$

$$\begin{aligned} V &= \sum_{i=1}^{N} v(s_i) \cdot 4^{N-i} \\ &= 2 \cdot 4^3 + 3 \cdot 4^2 + 3 \cdot 4^1 + 0 \cdot 4^0 \\ &= 188 \end{aligned}$$

Figure 4: When it is converted to an integer, the k-mer "GTTA" in $k = 4$ space maps to the integer value 188. Since this relationship is bijective, 188 also maps uniquely back to "GTTA".

Because the numeric values of the bases were assigned in lexical order (A = 0, C = 1, etc.), sorting k-mers in their numeric form is equivalent to sorting them as strings. When viewing the bases in their binary form (A = 00, C = 01, G = 10, T = 11), another useful property emerges: the complement of the base and its binary complement are the same. For example, $\text{A}^c = 11 = \text{T}$ and $\text{C}^c = 10 = \text{G}$.

Note that ambiguous bases, such as N, cannot be represented in this binary encoding scheme. Therefore, any k-mer containing an ambiguous base is undefined.

2.3.2 K-mer counting

Counting the unique k-mer frequency is a trivial task when the data are small. The simplest implementation would take each sequence read, k-merize it, and update a hash table in memory that maps k-mers to counts. When the number of k-mers is sufficiently small, this approach is fast and runs in $\mathcal{O}(n)$ time. When the data are large, caching all k-mers in memory is no longer feasible, and a different approach is needed.

One approach is to reduce k-mers from sequence read errors. Each one can produce up to k extra k-mers, and these will have a frequency around 1. One solution, as implemented in BFCounter [86], is to stochastically remove these singleton k-mers using a Bloom filter [10]. This statistical construct does not yield exact k-mer counts, but it does reduce the data. Another approach, as implemented by khmer [121], attempts to improve accuracy by using an array of Bloom filters. For large data, real k-mers can still overrun memory, and these stochastic filters cannot solve that problem. For our purposes, we will look at efficient implementations of exact counters.

DSK [102] is one interesting exact k-mer counter that implements a multi-pass approach over the sequence data using an impressively small amount of memory [4]. As a tradeoff, this implementation does not scale as well to larger data.

The most popular k-mer counter today is Jellyfish [80]. This program uses a limited-size hash table to store k-mer counts in memory. When the hash table reaches a set capacity, it is offloaded to disk. When all sequence reads are processed, the hash tables are merged. Jellyfish can fill hash tables using multiple threads, which will lead to race conditions when the same memory location is modified simultaneously. Atomic locks on the data structure can eliminate these conditions, but this approach would spend too much time acquiring and releasing the locks. Instead, Jellyfish uses compare-and-swap (CAS) operations to detect and recover from simultaneous updating. This advanced hash table implementation is very fast, and it has made Jellyfish the *de facto* choice for k-mer counting.

All of these implementations come with significant limitations. For example, the output of Jellyfish is a hashtable on disk that can be queried, but listing the set of k-mers is not as easily done. Therefore, comparing the k-mer distributions from two samples is not a simple algorithm since the records are distributed in the data structure according to a hash function. Custom transformations on k-mers are also not as easy to perform with these tools. Furthermore, none of them are distributed with a well-documented application programming interface (API), so they cannot be directly integrated into larger software applications.

For the algorithms described in later chapters, these existing tools had too many limitations, and so I created one that fit my needs.

2.3.3 KAnalyze

I set out to create not only an efficient k-mer counter, but an efficient toolkit of reusable parts. This work resulted in KAnalyze [4]. The project is designed around an API, which is then driven by a user interface or by another program, and a command-line user interface is included in the package. Other tools do not make the API available, so other programs must drive them by their user interface.

KAnalyze is a collection of components that can be connected in flexible ways. The only requirement is that the output type of one component match the input type of the next. Pipelines of components, or modules in KAnalyze, can be built in flexible ways. For example, a filtering component can be placed arbitrarily in a pipeline, and k-mers can be transformed at almost any step as needed. This flexibility has been critical in many of the applications and algorithms that follow.

Each component is connected by an in-memory queue that is synchronized. When no data is available for a component, it sleeps until an upstream component writes to the queue. The component then wakes up, processes the data, writes it to the next queue, and information continuously flows through the pipeline. Components run in threads for task-level paralellism, and multiple components may execute a single step for data-level parallelism. To avoid excessive overhead from synchronization, data is written in batches. Batch sizes and queue sizes can be configured, but are fixed at runtime so that the pipeline is memory stable. The value of KAnalyze lies with this pipeline structure because it is fast, flexible, and extensible.

2.3.3.1 Design philosophy

This project is written in Java for its flexibility and speed of development. Throughout the design and implementation of this software, the principals of scientific software engineering were observed [117]. KAnalyze comes packaged in a project under version control with Git and an automated build system managed by Apache Ant. Automated unit-test code is also included in the form of JUnit test suites, which can be easily executed using the build system.

Java is a strongly typed language, and so values and data structures have a defined format. This is perhaps the opposite of Python's "duck typing"[1], which defers all type checking to runtime. In Java, an API method can require that it be passed an object of a specific type, and it is up to the caller to make any transformations to that type. The interface of that type is known, and so there is no ambiguity about its structure or interface. In Python, however, the caller can pass anything to an API, including lambda functions. Python is an incredibly flexible and useful language, but it was unsuitable for an application like KAnalyze because everything is checked to mitigate bugs that are difficult to trace.

The disadvantage of using Java is that it lacks unsigned integers, and it severely limited performance when k-mers sizes were expanded past 31 (Section 2.3.3.3). However, the 31-mer limitation was debilitating for experiments on human data. This problem might have been solved by using other languages, namely C++, but other properties of Java still made it a good choice (see Section 2.3.3.2 for an example).

I believe that proper error handling is one of the cornerstones of good software design and that most programmers do not dedicate enough effort to it. For example, if KAnalyze reads a FASTQ file that contains irrecoverable problems, the error it reports includes what it found, what it expected to find, and the corresponding line number. I have seen too many bioinformatics applications that lack error detection so completely that it can't detect when it is given the wrong file type; the program either crashes with a cryptic error message, silently ignores the data, or inputs nonsensical

[1]Alex Martelli wrote in a newgroup, "In other words, don't check whether it IS-a duck: check whether it QUACKS-like-a duck, WALKS-like-a duck, etc, etc, depending on exactly what subset of duck-like behaviour you need to play your language-games with." (2000)

data. In a scientific setting, this should never be an acceptable practice.

Java makes it easy for the prudent programmer to write applications that are very difficult to crash, and the result is sometimes called "bomb-proof" code. These programs check everything, assume nothing, and even check for some cases that should never occur. For example, what if a FASTQ and FASTA are accidentally concatenated and used as input? The naïve parser would assume that it is still reading a FASTQ file when it encounters the FASTA records. It might generate "sequences" containing FASTA headers instead of sequence data. Other code, probably equally as naïve, might try to use this sequence and fail with some meaningless error. Since this crash would not immediately implicate the FASTQ parser or the input file, debugging this kind of code requires many unnecessary hours of user and programmer time. In contrast, the bomb-proof parser fully checks every record, and when it finds a problem, it reports the error. The user then knows which input file is to blame and where in the file the offending data was found. The naïve parser leaves the user with no information and no choice but to start troubleshooting blindly.

Figure 5 shows a sample of code from the FASTQ parser. This section of code is entered at the end of a sequence line or when whitespace is found while reading the sequence characters. It allows arbitrary whitespace at the end of the line, but it does not allow spaces embedded in the sequence. Lines 5 generates the error messages when these rules are violated, and line 15 generates an error when the end of the file is reached at this point, which would truncate the sequence record. The function `err()` reports the error and stops processing the file. `sourceName` and `lineNum` are the name of the input file and the current line number, respectively.

Built within KAnalyze is an event handling system that makes it possible for components and pipelines to report problems and make no assumptions about how they are handled. The events passed through this system are one of two classes. An *error* means the component aborted processing the data and that the analysis task

```
 1  // Only allow whitespace to end of line
 2  while (readBuffer[readBufferIndex] != '\n') {
 3
 4      if (! Character.isWhitespace(readBuffer[readBufferIndex]))
 5          err("FASTQ record contains text data after whitespace
                on a line with sequence data (whitespace may
                surround the sequence string, but may not be
                embedded in it): " + StringUtil.charDescription(
                readBuffer, readBufferIndex), sourceName, lineNum);
 6
 7      ++readBufferIndex;
 8
 9      // Refresh buffer
10      if (readBufferIndex == readBufferSize) {
11
12          readBufferSize = refreshBuffer(reader, readBuffer, 0);
13
14          if (readBufferSize <= 0)
15              err("FASTQ file truncated while reading sequence
                    data: End of file before reading quality data
                    for a record", sourceName, lineNum);
16
17          readBufferIndex = 0;
18      }
19  }
```

Figure 5: FASTQ parser code that is entered when whitespace is found in a sequence line after sequence characters are read. Arbitrary whitespace is allowed at the end of the line, but it may not be embedded in the sequence. Line 5 shows the error reported when this rule is violated. Line 15 shows the error reported when the end of the file is reached at this point in the FASTQ record, which would truncate the record before the quality string was read.

should be stopped. A *warning* reports conditions that were recoverable, but may have lead to incorrect results. When an event is reported, it is up to the implementation to decide how to handle it. For example, a command-line interface might write errors to standard output, but a larger system using the API might log it or stop some task the KAnalyze components are only a small part of.

Building this level of error handling into a system requires a significant development effort. KAnalyze dedicates thousands of lines of code to the error handling system and the checks that push events to it. In my experience, however, it takes less than 10 minutes to find and fix a bug, and this is only possible because error conditions are adequately reported. The time spent on error handling has saved countless hours of debugging and troubleshooting.

Documentation is also critical to making a project maintainable and usable. KAnalyze has a manual written in $\LaTeX$ and distributed as a PDF file. It contains example usage, all command-line options, and information about the structure of the KAnalyze project.

KAnalyze has a custom command-line processing library that allows each option to be defined with a single object, which contains the name of the option, default values, help text, and code for processing the option. Because options are defined in one place, it is easier to add and modify them. These options make API calls to configure the module, so applications using the API have access to all the same features.

For maintenance programmers and developers using the API, Javadoc comments appear on every class, method, and data member in the project. These comments can be used to build a navigable HTML site documenting all elements of the project. Documentation generation is automated in the build system.

This section describes good software engineering principals that helped make this project a success. As part of my work, I hope to inspire others bioinformatics programmers to use them.

2.3.3.2 Dynamically loadable classes

Many features are implemented by classes that are loaded dynamically at run time. These classes must implement an interface known to KAnalyze and have a default constructor. Java's Reflections API and flexible class-loading system make it possible to find and load the classes at runtime. Custom components can be created by a user, compiled independently from the rest of the code, and loaded at run time. KAnalyze needs to know nothing about these components *a priori*.

Several features are handled by these dynamic elements. For example, the sequence reader component uses the input type, such as FASTA or FASTQ, to find and load a class that handles converting the bytes in the input files into batches of sequence reads. Because of this architecture, a custom reader can be integrated directly into KAnalyze. It can parse sequence data in any format, and it could even read from a database. KAnalyze can automatically resolve the reader for custom file patterns and even output help on the command-line for custom readers as if the reader were part of the KAnalyze system. At its core, custom readers and readers shipped with the package are not treated any differently; they are all dynamically loaded as needed.

Like the sequence reader, the output writer follows the same convention. The desired format can be specified on the command line, and the writer is automatically loaded at run time. These also have methods for getting information about the writer, so that KAnalyze can list them and a help string on the command-line.

Filters, discussed in Section 2.3.5, are also loaded dynamically, and these enable a vast array of custom transformations that can be applied to data within the KAnalyze pipeline.

This is one of the features that sets KAnalyze apart from other k-mer processing programs. Instead of a narrowly defined use case, KAnalyze can be shaped to meet the unique needs of applications that I, the original programmer, could never imagine. As I developed the algorithms and software described in later chapters, I made full use of this flexibility.

2.3.3.3 K-mer structure

In the KAnalyze pipeline, binary k-mers are packed into arrays of 32-bit integers each encoding 16 bases, and each 32-bit integer is referred to as a *k-mer word*. The binary representation of k-mers is described in Section 2.3.1.

Integers are two's complement where the highest-order bit is set when the number is negative, and there is no support for unsigned integers or alternative primitive data structures in Java. This presents a challenge for comparing k-mer order, and KAnalyze must sort k-mers to count them (Section 2.3.4). If a k-mer word starts with G or T, its integer representation is negative, and it would be sorted before a word starting with A or C. There are several approaches to solving this problem, such as comparing a copy of the word with the highest-order bit negated, but the copy operation caries a significant performance penalty.

An alternative approach is to test for negative numbers and handle them accordingly. Algorithm 1 is an example of how this is implemented in KAnalyze to test for k-mers for a strictly greater-than relationship. To further mitigate the performance penalty, shortcut logic evaluation is used once the sign of one k-mer is known.

Algorithm 1: Greater-than k-mer comparison with 32-bit signed integer words

```
Input: $left$: Left k-mer as an array of 32-bit words
Input: $right$: Right k-mer as an array of 32-bit words
Input: $N$: Number of words per k-mer
Output: A boolean value indicating TRUE if $left$ is strictly greater than $right$ and
        FALSE otherwise
1 $index \leftarrow 0$                                    ▷ Loop control variable
2 while $index < N$ do
3     if $left[index] \neq right[index]$ then
4         if $left[index] < 0$ then
5             return $right[index] \geq 0$ || $right[index] < left[index]$
6         else
7             return $right[index] \geq 0$ && $right[index] < left[index]$
8     $index \leftarrow index + 1$
9 return FALSE
```

It is possible that performance may be improved by simply avoiding the high-order and allocating 15 bases per word instead of 16. Within memory, this makes sense. It would require more memory and disk space when k-mers are offloaded as binary structures. Also, not all programming languages suffer from a lack of unsigned integers, and this would make little sense in that context. Complicating the data structure may also complicate future applications. Here I made a design decision to suffer the performance impact for comparisons and keep the structure cleaner.

2.3.3.4 K-mer algorithm

As outlined in Section 2.2.1, a sequence is converted to a k-mer using a sliding window of size k over the sequence. Because neighbors share all but one base, recomputing the whole window at each step is unnecessary. To find the next k-mer from the previous one, the whole k-mer is shifted by one base, which discards the first base, and the next base from the sequence is appended. This process continues until the last base is read. If an ambiguous base is encountered, such as N, the current k-mer must be discarded and rebuilt starting with the next base.

Algorithm 2 depicts the process of converting the sequence read. For the purpose of illustrating this algorithm, the k-mer words are concatenated into a single long word, *kval*, which has exactly the number of bits needed to represent the k-mer. When it is shifted to the left by two bits, the left-most base is discarded, and two zero bits are inserted on the right end. *load* keeps track of how many valid bases are in *kval*, and it is reset when an ambiguous base is encountered. Only full k-mers are emitted.

KAnalyze implements Algorithm 2 without a switch statement. Instead, an array is pre-loaded with the numeric values for each base character to avoid the multiple tests and jumps associated with the switch implementation.

2.3.4 KAnalyze count

The KAnalyze count module outlined in Figure 6 breaks k-mer counting into two distinct phases. The first phase is executed by all elements up to the split component, and the second phase begins with the merge component. With this parallel pipeline structure, KAnalyze is able to scale to handle large amounts of data.

In the first phase, k-mers are accumulated into a memory array up to a configurable

```
Algorithm 2: String to numeric k-mer conversion
    Input: $s$: A nucleotide sequence
    Input: $k$: Kmer size
    Output: A set of numeric k-mers
 1  $M \leftarrow \emptyset$                                      ▷ Multiset of k-mers
 2  $kval \leftarrow 0x0$                                         ▷ Current k-mer
 3  $load \leftarrow 1$                        ▷ Number of valid bases in $kval$
 4  for each character $c$ in $s$ do
 5      $kval \leftarrow$ lshift($kval$, $2$)    ▷ Left shift 2 bits (discards first base)
 6      switch $c$ do
 7          case $A$ do
                                                  ▷ Nothing to add:  A is 0
 8          case $C$ do
 9              $kval \leftarrow kval \mid 0x01$
10          case $G$ do
11              $kval \leftarrow kval \mid 0x02$
12          case $T \mid U$ do
13              $kval \leftarrow kval \mid 0x03$
14          otherwise do
15              $load \leftarrow 0$
16      if $load = ksize$ then
17          $M \leftarrow M \cup kval$
18      else
19          $load \leftarrow load + 1$
20  return $M$
```

limit. The array is sorted, the number of occurrences of each k-mer is counted, and this set of sorted k-mers and counts is offloaded to disk. Each of these groups of k-mers is called a “segment”. The array is then filled with the next set of k-mers, and the process repeats. To reduce disk space consumption by segment files, binary k-mers are written, and k-mer counts are compressed using a variable-length ULEB-128 encoding.

The second phase merges all segments using a modified external merge sort algorithm [56], and the final k-mer counts are sent to output.

Sorted output is inherent to this k-mer counting approach, and this has some useful properties. When comparing the k-mers of two or more samples, a trivial

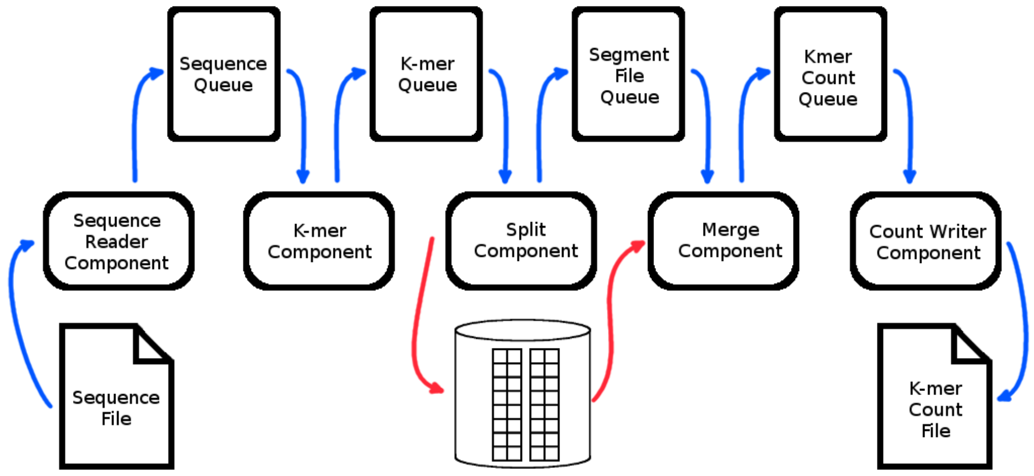

Figure 6: The count pipeline contains 5 basic steps. Blue arrows depict data transfer in memory, and red arrows depict data transfer to disk. Sequence data is read from a file, batched in memory, and sent to a component that translates the sequences to binary k-mers. The split component aggregates a set number of k-mers in memory, sorts them, and offloads the k-mers and counts to disk. When all sequence data has been processed by the split component, the merge component merges the k-mer counts from each file. Finally, the k-mer counts are written to output.

$\mathcal{O}(n)$ algorithm can read the k-mers for each set. This sort order can also be defined arbitrarily, which was a key to efficiently building a new data structure essential for quickly randomly querying k-mers from disk, as discussed in Section 2.3.6.

2.3.5 Filtering and transformations

Because of the pipeline structure, filtering and transformations can occur in a number of places without modifying existing components. With dynamic component loading (Section 2.3.3.2), this is one of the most powerful features of KAnalyze.

2.3.5.1 Filtering k-mers

The count pipeline has hooks for dynamic k-mer filters in two places: after k-mers are generated from sequence data (*pre-count*), and after they are counted (*post-count*).

The pre-count filter can remove or modify k-mers before they are aggregated and counted. Typically, k-mers are reverse-complemented or changed to their canonical form at this step (Section 2.2.2). Section 2.3.2 discusses stochastic k-mer counting, and this feature could be added to KAnalyze by placing a filter at this location.

The post-count filter is typically used to filter low-frequency k-mers. This was the goal of the stochastic k-mer counters, however, KAnalyze can accomplish the same goal with exact k-mer counts. The stochastic filters may still be useful for speeding up the count pipeline in particularly time-sensitive tasks, but this remains untested.

These filters are unrestricted, and they can alter k-mers in any way needed to accomplish a task. For example, KAnalyze has an option to output the forward and the reverse complement of each k-mer, and this pre-count filter writes twice as many k-mers as it reads.

2.3.5.2 Sequence reads

The overall quality of k-mers can be improved by removing low-quality bases, and FASTQ files encode this information for each base. When enabled, the count pipeline will automatically attach a quality filter to the sequence reader. This filter examines each base's quality score, and when it is below a set threshold, the corresponding base in changed to an N. Because ambiguous bases cannot be represented (Section 2.3.1), all k-mers containing N are discarded. This effectively removes all k-mers that would have contained a low-quality base.

All sequence readers use filters, but only the FASTQ quality filter is packaged with KAnalyze. Custom filters for any reader may be defined and loaded at run time.

2.3.6 The indexed k-mer count format

2.3.6.1 Minimizers

Minimizers [103] are a method for grouping k-mers such that neighboring k-mers in a sequence read are likely in the same group, and this arrangement can be exploited to create fast random access data structures by selectively loading subsets of k-mers into memory as needed. Kraken [118], a metagenomic application, uses this method to rapidly query phylogenetic groups associated with k-mers.

To find the minimizer of a k-mer, all forward and reverse-complement sub-k-mers of a set size are extracted. The lesser of these sub-k-mers, as defined by sort order, is the minimizer of the original k-mer. The next k-mer in a sequence will likely have the same minimizer because neighbors overlap by $k - 1$ bases. Therefore, grouping k-mers by their minimizer and sorting within the groups creates a data structure such that once one k-mer is found, the same group can be searched for the next k-mer.

The advantages are realized when this minimizer-grouped data structure resides on disk and is opened as a memory mapped file. Pages of this file are swapped into memory as they are accessed, and they are swapped out of memory when no longer needed. Therefore, once a k-mer is searched, the whole minimizer group is likely loaded into memory for the next k-mer search. This creates a data structure that can be queried efficiently without loading the whole set into memory.

The disadvantage to this approach is that it is relatively expensive to compute minimizers. For this reason, the minimizer of a k-mer is computed only if it is not found in the last minimizer group. If the k-mer is not found a second time, then it does not exist in the file. Kraken uses this same optimization.

KAnalyze is organized so that all k-mer comparisons are implemented by an interface, and the class implementing that interface determines the sort order. When output is written to an IKC file, or if minimizers are turned on, KAnalyze uses an implementation of the class that tests the minimizer first and then the k-mer if the minimizers are equal.

Sorting with a minimizer suffers a massive performance penalty if the minimizer is computed for each comparison. Since k-mers are implemented as an array 32-bit words, an additional word is allocated for each k-mer to cache the minimizer. To simplify comparisons, the maximum minimizer size is 15, which occupies 30 bits. This conveniently avoids the high-order bit, and so minimizers can be compared without testing for sign.

2.3.6.2 IKC file format

```
   |    1    |    2    |    3    |    4    |    5    |    6    |    7    |    8    |
---+-------------------------------------------------------------------------------+
 1 |                                     MAGIC                                     |
---+-------------------------------------------------------------------------------+
 2 |                                     MAGIC                           | VERSION |
---+-------------------------------------------------------------------------------+
 3 |                              RESERVED                               | MIN_SIZE|
---+-------------------------------------------------------------------------------+
 4 |               KMER_SIZE               |                  MASK                 |
---+-------------------------------------------------------------------------------+
 5 |                                  INDEX_OFFSET                                 |
---+-------------------------------------------------------------------------------+
 6 |                                  META_OFFSET                                  |
---+-------------------------------------------------------------------------------+
 7 |                                      ID                                       |
---+-------------------------------------------------------------------------------+
 8 |                                      ID                                       |
---+-------------------------------------------------------------------------------+
 9 |                                      ID                                       |
---+-------------------------------------------------------------------------------+
10 |                                      ID                                       |
---+-------------------------------------------------------------------------------+
   |                                                                               |
   |                                                                               |
   |                                     DATA                                      |
   |                                                                               |
   |                                +------------+                                 |
   |                                | KMER | KEY |                                 |
   |                                +------------+                                 |
   |                                                                               |
   |                                                                               |
   |                                                                               |
---+-------------------------------------------------------------------------------+
   |                                                                               |
   |                                    INDEX                                      |
   |                            +--------------------+                             |
   |                            | MINIMIZER | OFFSET |                             |
   |                            +--------------------+                             |
   |                                                                               |
---+-------------------------------------------------------------------------------+
   |                                                                               |
   |                                   METADATA                                    |
   |                                                                               |
   |                                +-------------+                                |
   |                                | KEY | VALUE |                                |
   |                                +-------------+                                |
   |                                                                               |
   |                                                                               |
---+-------------------------------------------------------------------------------+
```

Figure 7: Structure of an IKC file.

The KAnalyze API includes classes for reading and writing this data structure, which it calls the *indexed k-mer count* (IKC) format. To output this file efficiently, k-mers must be received by the IKC writer in their minimizer groups, and k-mers in each group must be sorted.

The IKC format is shown in Figure 7. The file header starts with a magic value, which all IKC files must have. This is a NULL-terminated string containing "Idx_Kmer_Count" (UTF-8/ASCII encoded). The next field is a version, and it tells the IKC reader how to interperet the file. The format shown here is version 1. Reserved fields are strategically placed for future use. The k-mer size is next. A minimizer mask in an XOR mask applied to minimizers to break up large groups due to low complexity, such as repetitive "A" in the sequence data. By default, this is 0 (binary string of all zeros), which disables masking. The index and metadata offsets are 64-bit file locations for these sections, which are described below. Lastly, the ID field is filled arbitrarily by the writer, and it may use the field to identify what software created the file.

The data section begins immediately after the header, and it is an array of k-mer records. Each record is allocated enough bytes to store the k-mer in its binary form. Each k-mer associated value, or key, which is a signed 32-bit integer. The KAnalyze count pipeline writes the frequency into this key, but the key may be any value as needed by the application. These records are sorted by minimizer and then by the k-mer.

The index section is an array minimizers and the file offset where the minimizer group begins, which will always point to a location in the data section. Each record is a 32-bit integer and a 64-bit offset.

The metadata section follows the index, and it is optional. Instead of the data key representing a k-mer count, it can represent a key in this metadata section. Therefore, the key is a 32-bit integer. The value is a null-terminated string, and it is up to the implementation to define the format of that string. This section allows the k-mers to index any kind of data, such as phylogenetic associations. The IKC format was built with applications like Kraken in mind, but it was made flexible enough to associate k-mers with any kind of data a particular application might demand.

2.3.6.3 IKC Searching

When the KAnalyze IKC reader loads a file, it first reads the index section and builds a binary tree so that the start and stop offsets of each minimizer group can be looked up in $\mathcal{O}(\log n)$ time.

When looking up a k-mer, it first searches the last minimizer group with a binary search. If the k-mer is not found, the minimizer is computed, and if it is different from the last group, the correct minimizer group is searched. If the k-mer is not found on the second search, the key returned is 0. These are the optimizations that Kraken described.

Because k-mers from the same sequence tend to be grouped by minimizer, the minimizer group is loaded into memory when the IKC is opened as a memory mapped file. When a k-mer group is not used, its memory is freed. With this format and the search optimizations, the IKC provides very fast k-mer lookups using minimal resources.

2.4 Results

2.4.1 Methods

Testing was executed on a 12 core machine (2 x Intel Xeon E5-2620) with 32 GB of RAM (DDR3-1600), RAID-6 over SATA drives (3 GB/s, 72K RPM), and CentOS 6.7. Commands were executed with custom C code that spawns the process, restricts it to 4 cores, and monitors its memory usage. 2 cores were chosen from each processor on this machine, and they were not part of the same hyper-threading group.

Two data sets were tested independently, human chr1, from the hg19 reference, and SRR014079, a random 1000-Genomes data set. These were chosen because they contain a similar amount of sequence. However, chr1 is an assembled sequence in a FASTA file, and SRR014079 is a set of sequence reads in a FASTQ file. Both represent a potential use of KAnalyze that is different. Subtly, chr1 also tests the KAnalyze sequence readers, which can break large records into fragments that produce the same k-mers as the whole sequence. This preserves memory, and the parallel pipeline does not have to wait for a large sequence to be loaded.

KAnalyze and Jellyfish were tested with 31-mers and 128-mers. Both of these applications were originally restricted to 31-mers, but later updates added large k-mer support. 31-mer performance is measured to preserve historical information, and 128-mers are tested to see how well the program scales as the k-mer size is increased.

For KAnalyze, the output is a sorted tab-delimited text file of k-mers, as nucleotide strings, and their counts. The *Java virtual machine* (JVM) is allowed 3 GB, and all default KAnalyze options were used. A pre-2.0 release of KAnalyze, 1.0.0dev5, was used for these tests

Jellyfish parameters need to be tuned to maximize performance given the resources available. Before executing these tests, preliminary benchmarking was done to determine the best parameters for running in 3 GB of RAM and 4 cores. For 31-mers, Jellyfish was run with a hash size of 536870912 elements and 4 threads. For 128-mers, Jellyfish was run with a hash size of 134217728 and 4 threads. Jellyfish version 2.2.6 was used for these tests.

Jellyfish outputs a hash table of k-mer counts. The records are in an order defined by the hash function, which is effectively random. Jellyfish can write k-mers to a FASTA file where the record ID is the k-mer count. Custom Python code converts that FASTA output to a sorted tab-delimited k-mer count file, identical to KAnalyze output, so that the results can be compared. This conversion is not included in the Jellyfish runtime or memory metrics reported in this section.

A custom Python pipeline executes k-mer counting with a simple naïve algorithm, and it reflects what an average bioinformatician would do if all k-mers could not be loaded into memory as they are counted. Each sequence is read, k-mers are extracted as substrings, and each is dumped into a file. The file is sorted with the Linux `sort` command. Python code then reads the sorted file and collapses the list of k-mers into k-mer counts. The output produced is identical to the KAnalyze output.

The final tab-delimited k-mer count file from each of these three approaches (KAnalyze, Jellyfish, and the simplistic Python script) should be exactly the same if each approach is correct. A SHA-1 checksum is computed over each file, and the checksum for each is compared.

2.4.2 CPU performance

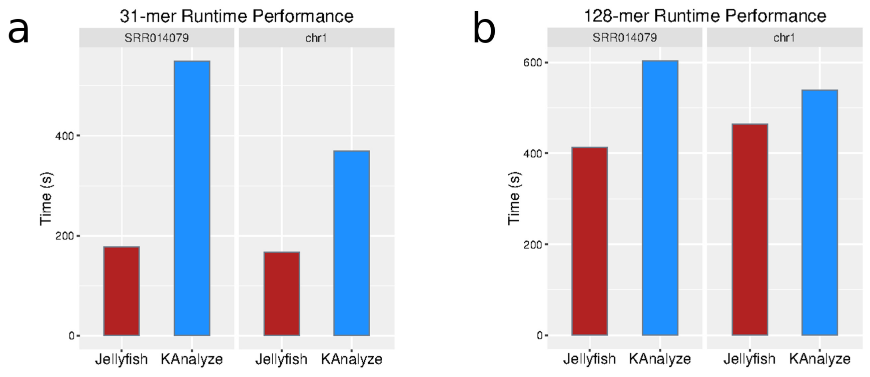

Figure 8: KAnalyze runtime performance vs Jellyfish. (**a**) 31-mer (**b**) 128-mer.

For 31-mers over both data sets, Jellyfish is clearly faster than KAnalyze (Figure 8(a)). For 128-mers, however, KAnalyze begins to close the performance gap, and this suggests that is some cases, KAnalyze may scale better (Figure 8(b)).

The original KAnalyze publication [4] tested the same data sets, but in that case, KAnalyze was faster than Jellyfish. This loss of performance in the current version is due mostly to the fact that k-mers are compared by 32-bit words, and the words have to be compared accounting for negative numbers because Java does not have unsigned integers (Section 2.3.3.3). The original version restricted k to 31 or less, and the k-mer was stored as a 64-bit long integer.

Despite these differences, it would still likely take far more time to convert the

Jellyfish output to some of the formats that KAnalyze can output natively, such as the IKC file. Recall that the Jellyfish output is an unsorted hash table and that the time to perform conversions to the k-mer count format are not included in these metrics. The conversions added an average of 2707 seconds to the 31-mer SRR014079 test case and 2778 seconds to the 31-mer chr1 test case (approximately a 5x KAnalyze speedup when compared to Jellyfish).

2.5 Discussion

The flexibility of KAnalyze makes it a powerful tool for transforming sequence data. Its pipelined architecture enables non-trivial data transformations with both built-in and custom components. Filters can be inserted in almost any step, and k-mers can be altered in any imaginable way. Furthermore, KAnalyze's merge-sort approach to counting not only sorts the output, but it makes it possible to write more complex data structures, such as the IKC format. KAnalyze provides a foundation for k-mer transformations unlike anything available today.

Jellyfish is the *de facto* k-mer counter, but since it is very rigid in its design, it cannot be extended easily. Since its output is a hash table of k-mer counts, the results must be loaded into memory or transformed to a different data structure. The flexibility of KAnalyze makes it amenable to new ways of using k-mers. KAnalyze cannot compete with Jellyfish in terms of raw speed to its native output, however, it can better support many algorithms where a hash table is not practical.

The effort to build this platform was a software engineering endeavor. If scientific programs are unstable, then the inferences made from their results are unreliable. Error conditions, no matter how unlikely, must tested before they cause the program to crash or output incorrect data, and well designed software testing practices should be used to ensure the stability of the code. As a scientific software engineer, this is an issue I take seriously, and I hope to inspire others to do the same.

CHAPTER III

ALIGNMENT-FREE SPOLIGOTYPING

3.1 Abstract

Tuberculosis (TB) is a global threat that claims more more than a million lives each year. Fortunately, new treatment options and better monitoring capabilities have led to a sharp decline in TB disease over the past few decades, and rates continue to fall. Although these improvements have saved many millions of lives, additional advances are needed to reach the World Health Organization (WHO) goal to end the epidemic by 2035.

Some of the most powerful new methods use whole genome sequencing (WGS) to characterize infections and outbreaks. With advanced techniques, this data can precisely determine the lineage of the bacteria and accurately identify patterns of drug resistance. WGS is a powerful tool in the battle to end TB, but advanced analysis methods are needed to maximize its impact.

A great deal of information is still stored in the context of older techniques, such as spoligotyping. As the transition is made from the laboratory assays to modern WGS methods, it is helpful to use sequencing data to recapitulate the results that would be obtained from *in vitro* experiments. This chapter discusses a new method for performing spoligotyping from sequencing data.

As is common in scientific endeavors, the answer to one question inspires a new question. This was my experience with this work, and in trying to find a viable solution to one problem, I realized that it had much broader implications. This led directly to the work presented in CHAPTER IV.

3.2 *Introduction*

3.2.1 Treating Tuberculosis Disease

According to the 2015 World Health Organization (WHO) Global Tuberculosis Report [1], 9.6 million people were ill with Tuberculosis (TB) disease in 2014, and 1.5 million of those victims died. TB ranks with HIV/AIDS, which claimed 1.2 million lives in 2014, as the leading cause of death by infectious diseases. Multi-drug resistant (MDR) cases are of particular concern, which accounts for 3.3% of new cases and 20% of previously treated cases. Of the MDR cases, 9.7% are extensively drug resistant (XDR), and these are far more difficult to treat.

Fortunately, 43 million lives have been saved since 2000, and TB mortality has fallen 47% since 1990 [1]. In the past decades, better treatment options and diagnostics have changed the landscape of this disease. The WHO intends to end the TB epidemic with a targeted 95% reduction of deaths between 2015 and 2035. With current tools and trends, the annual decline in the TB incidence rate is 1.5%, however, a 10% annual decline is needed by 2025 to reach the 2035 goal [95] (Figure 9).

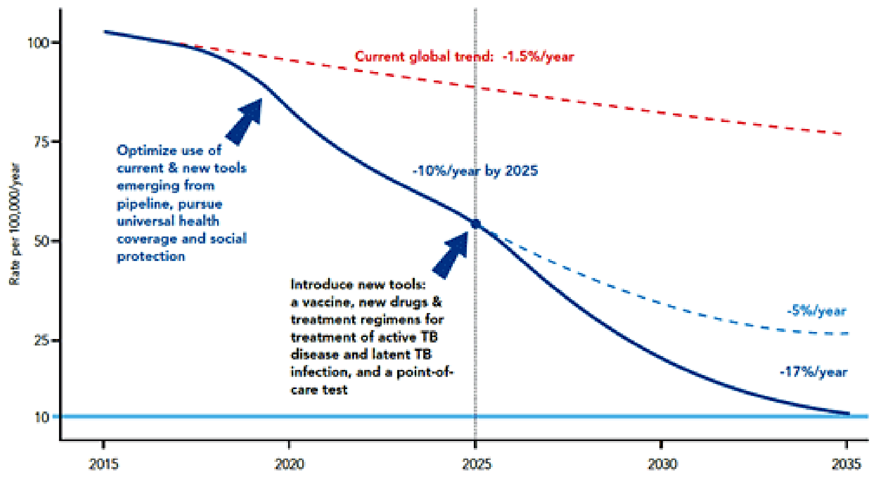

Figure 9: The annual decline in TB incidence must drop from 1.5% per year to 10% per year by 2025 to reach the goal to end the TB epidemic by 2035. Image retrieved from the WHO website [95]

As our understanding of TB improves, agile analysis techniques are critical for turning it into actionable information. Strain type and drug resistance profiles, currently based on laboratory assays, are two of the most important. Today, many of these are still *in vitro* assays performed in a laboratory setting. While these assays continue to improve, they are still rigid in their application. For example, the line probe assay (LPA) is a method that uses PCR to amplify specific resistance regions, and it detects drug resistance with labeled probes far more rapidly than traditional drug susceptibility tests [25]. Updating LPA for new drugs, such as bedaquiline and delamanid, would require designing new primers and probes. However, since it is one of the best methods today, the WHO recommends its use [1].

Typing is the focus of this chapter, but there is significant overlap with types and drug resistance. It is well documented that lineage 2, known as the *Beijing lineage*, is associated with higher rates drug resistance and MDR TB cases [11, 32]. A higher rate of mutation in this lineage appears to be responsible for its ability to acquire resistance to many anti-TB drugs [37] (Figure 10). Rapidly determining the lineage and other genetic properties of samples can have a profound impact on how infections and outbreaks are to be treated.

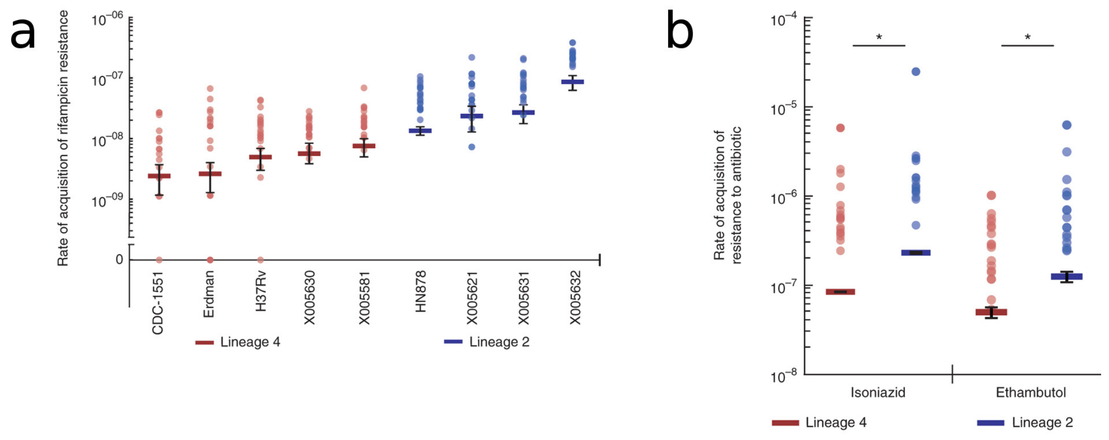

Figure 10: The TB Euro-American lineage (4) and East Asian lineage (2) acquire drug resistance at different rates. (**a**) rifampicin. (**b**) isoniazid and ethambutol. Images published by Ford *et al.* [37]

3.2.2 Bacterial typing

Bacterial typing was traditionally based on *in vitro* assays, and these methods are useful for broadly grouping samples. Modern typing can detect outbreaks, and WGS analysis can reveal transmission chains within an outbreak [57, 17]. This is critical information for anyone attempting to control an infectious disease.

Separating a bacterial species into groups is an original goal of bacteriology, and it reveals useful clues about species evolution and epidemiological surveillance. These groups tend to share important characteristics, such as ancestry, virulence, and resistance to antibiotics.

The traditional phenotype methods measure characteristics of the sample, but they do not directly measure the genome. Typing by the differential lytic activity of phages is one of the earliest. Bessie Callow was the first to observe that isolates from separate infections reacted differently to phages [14]. By 1953, this method was well established for *Staphlycoccus spp.* [9], when it was capable of identifying four distinct groups with known antibiotic susceptibility statistics for each.

Serotyping is a method of using antibody specificity. It was developed by Rebecca Lancefield in 1933 for typing *Streptococcus spp.* [62], and it is still used in clinical settings [114].

These phenotypic methods still have relevant applications today, however, they have some critical limitations. For example, phage typing sometimes groups unrelated individuals and fails to group related ones [114]. With modern technology and our understanding of bacterial genomics, more powerful genotypic methods are becoming available.

M. tuberculosis typing methods currently in use include phage typing, drug susceptibility profiling, IS6110-restriction fragment length polymorphisms (RFLP), variable number of tandem repeats (VNTR), and spoligotyping [108]. Even though modern genotyping techniques are more sensitive, some of these methods are still capable of at least differentiating outbreaks, and there is valuable historical data stored in the context of these types. For example, when spoligotyping the notorious Beijing subtype of the East Asian lineage, samples hybridize with spacers 35-43 [32]. As the field shifts toward newer genomic methods, there must be a way to recapitulate this information for new samples without performing additional expensive laboratory assays. Fortunately, some of them can be determined by examining the genome.

Spoligotyping characterizes clustered regularly interspaced short palindromic repeat (CRISPR) region of the genome, which contains many unique spacers each separated by a short direct repeat sequence. This region is part of a bacterial adaptive immunity system designed to stop viral infections [6], and the unique spacers contain a record of past viral infections that are heritable by sister cells when a bacterium replicates. Spacers are also sometimes lost by deletion events within the CRISPR locus. The gain and loss of these unique spacers occurs at a rate sufficient to make identifying them useful for typing.

Spoligotyping was originally a hybridization assay designed to simultaneously detect and type *M. tuberculosis* species [55]. With membrane-bound oligonucleotides and tagged primers, this assay amplifies the CRISPR genomic region and detects the presence of 43 specific spacers. In 2000, additional spacers were characterized [30]. It is used in tandem with other methods today [43, 115], and it is still part of pathogen surveillance efforts at the Centers for Disease Control and Prevention (CDC).

3.2.3 Sequencing TB

When treating an individual, which tests should be run? Should typing be routine, and if so, at what resolution? Which drug-resistance variants should be detected? Fortunately, genome sequencing can replace many of these tests with one laboratory procedure and many *in silico* assays on the resulting data. Furthermore, when important new patterns are confirmed, they can be instantly distributed electronically.

As sequencing technology becomes more affordable (Section 1.2), this technology becomes more widely available.

The need to culture isolates makes sequencing many days slower [58, 16]. Fortunately, it was recently shown that MTBC cultures can be extracted directly from BACTEC™ MGIT™ tubes for sequencing in as little as three days [116], and metagenomics approaches may be a way to bypass culturing altogether [26]. As sequencing becomes a more viable approach, automated analysis techniques must be made available [58].

While traditional typing methods are useful for linking isolates to a particular outbreak, they lack the precision to construct transmission chains among individuals in an outbreak. For this work, WGS is often applied to find the source of individual infections and identify super-spreaders [57, 17]. *M. tuberculosis* is not naturally competent, and so only a few SNPs may differ among individuals in an outbreak. It is also a slow-growing bacteria, and so extracting and amplifying small amounts of DNA can lead to a result faster than many traditional laboratory assays.

As the barriers for TB sequencing are lowered, so is its cost. With benchtop sequencers, there is no need to send a sample to another location, and the time to generate sequencing data can be further reduced [77, 48]. The future of clinical microbiology depends on WGS analysis [8], and this technology is improving at just the right time to make the WHO 2035 goal a reality.

3.2.4 K-mer spoligotyping

The CDC *M. tuberculosis* science team, which is led by Dr. Jamie Posey, asked me to apply KAnalyze [4] to spoligotype and characterize spacers by looking at sequence data as k-mers sets. They were using existing software, Lasergene (DNASTAR®, Madison, WI) to perform spoligotyping with DNA sequence data amplified over the

CRISPR region. This software would align the sequence reads with the spacer sequences and estimate the sequencing depth for each spacer. The team was not satisfied with the results because it took 30 to 60 minutes to run on a plate of 96 samples, and when it ran low on memory, it appeared to give erroneous results.

Using the KAnalyze API, I developed a spoligotyping application to read sequence data and estimate the read depth of each CRISPR spacer. This resulted in a much faster and much more reliable application.

While developing this software, I realized the importance of correcting for SNPs and small indel variants in the samples. Because the sequences detected by spoligotyping are short, and because k-mers are rigid structures (Section 2.2.5), a single variant could cause a significant portion of the data to disappear. I was able to find a solution that worked well for this problem, but it would not generalize well to larger sequences. It was only after I completed this project that I was able to solve this problem for a more general case. That work deserves special attention, and so it is presented in CHAPTER IV.

3.3 Methods and algorithms

3.3.1 Data

From Dr. Posey, I obtained IonTorrent sequence data amplified over the CRISPR region for 92 samples and sequences for 68 CRISPR spacers with lengths between 29 bp and 43 bp (median = 38.0 bp).

3.3.2 Determining spacer depth

The approach described here uses k-mers counted from the sequencing sample. Because the CRISPR spacers are short, a k-mer size of 16 was chosen. All k-mers in the sample are counted using the KAnalyze API, low frequency k-mers are filtered out, and the resulting frequencies are stored using a hash set in memory. This project was completed with the original KAnalyze implementation, which did not have some of the advanced features discussed in CHAPTER II. The indexed k-mer count (IKC) format had not yet been implemented, and so the original version of this software had to keep k-mers in memory for random access.

The k-mers of the spacer sequence are determined, but they are left in order. The k-mer frequencies from the sample are assigned to the k-mers in the spacer sequence. To avoid biases in spacer calls, the minimum frequency is the estimated read depth of the k-mer.

3.3.3 Sequence variants

Within the 92 samples, 13 contained a single nucleotide polymorphism (SNP), and 2 contained a single-base deletion in a spacer sequence. Since all k-mers that cross a variant are different from the reference, the expected frequency for these reference k-mers is 0. A single SNP can alter up to k k-mers since that altered base will appear at each position of the k-mer. With a median spacer length of 38.0 bp, a significant number of k-mers would differ between the reference and the sample. These variants have to be accounted for in order to obtain reliable results.

When a sample contains a variant within a CRISPR sequence, there will be a clear disruption of the distribution of k-mer frequencies over that region (Figure 11, red line). If at least one k-mer in the spacer is not altered, then it can be used as a seed for a correction attempt. Assuming that the variant is a single SNP, shifting the high frequency k-mer over one base will reveal which base is at that location in the sequence data. This is done by removing a base from one end and trying all four nucleotides at the other end.

With this method, the altered base can be found and corrected, but the algorithm cannot determine why the base was altered. That base could a SNP, and in that case, the variant is easily corrected by accounting for it at one locus. That base could also be the start of an insertion or a base following a deletion. If the variant is

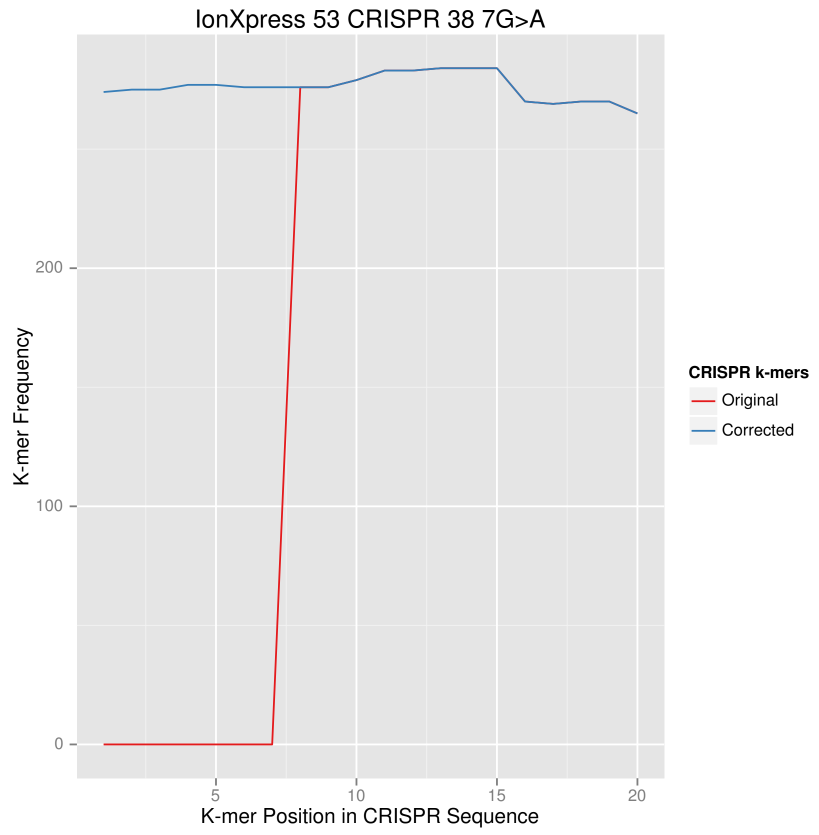

Figure 11: An example of variant correction in one sample with a variant in a CRISPR spacer. The red line represents k-mer frequencies for the uncorrected spacer sequence, and the blue line represents k-mer frequencies for the corrected spacer sequence.

an insertion or a deletion, then it must be corrected accordingly, and the remaining sequence must be shifted to remove a base or to make space for an inserted base. Since the reason for the alteration is not known *a priori*, one correction attempt will change one unaltered state into 3 altered states and 1 unaltered state. All these possibilities must be explored independently. Because more than one variant can occur in the sequence, this is a $\mathcal{O}(4^n)$ problem, and it will not scale to large sequences. However, the CRISPR spacers are relatively small, and so these paths can be explored quickly.

Algorithm 3 summarizes the correction process. A "call state" data structure is created assuming the original sequence contains no variants, and it is assigned as the

Algorithm 3: Correcting errors using k-mer frequencies. This illustration assumes that errors are corrected from left to right.

```
Input: $c$: An ordered list of k-mer counts in a CRISPR sequence
Input: $max$: Maximum number of corrections allowed
Input: $threshold$: K-mer count difference that triggers a correction attempt
Output: A call state that represents the corrected sequence
Data: $C$: A structure containing the current call state
Data: $S$: A set of call states
Data: $M$: Call state with the highest minimum k-mer count
Data: $index$: Current index in $c$
1  $S \leftarrow \emptyset$
2  $C \leftarrow$ NewCallState($c$)
3  $M \leftarrow C$
4  while $S \neq \emptyset$ do
5      $C \leftarrow S_1$                                  ▷ Get next state
6      $S \leftarrow S/C$                                  ▷ Remove state from $C$
7      $index \leftarrow C.index$
8      if Min($C.c$) > Min($M.c$) then
9          $M \leftarrow C$        ▷ Current state has the highest minimum k-mer count
10     while $index <$ Length($C.c$) do
11         if $(C.c_{index} - C.c_{index+1}) > threshold$ then
12             if $C.corrections < max$ then
13                 $S \leftarrow S \cup \{$ CorrectSnp($C$, $index$) $\}$
14                 $S \leftarrow S \cup \{$ CorrectIns($C$, $index$) $\}$
15                 $S \leftarrow S \cup \{$ CorrectDel($C$, $index$) $\}$
16         $index \leftarrow index + 1$                    ▷ Move to the next element of $c$
17 return $M$
```

optimal state until a better one is found. This call state contains all the information necessary to keep track of the sequence, k-mers of the sequence, frequency of those k-mers, and the current position in the sequence where scanning and correcting should continue.

The k-mers of the state are traversed until a large difference in the frequency of neighboring k-mers is found. It then attempts to correct the sequence as if it is a SNP, an insertion, and a deletion. Each case creates a new call state with the sequence and k-mers updated, and each state is added to a set of states. The algorithm continues with the next k-mer in the current state as if the frequency difference was erroneous.

When the end of the sequence is reached, it is abandoned and the next state is taken from the set. Each time a call state is taken from the set, it is checked against the optimal state, and if it has a higher minimum k-mer frequency, then it becomes the new optimal state.

The optimal state when the set of possible states is exhausted is returned by the algorithm. The minimum k-mer frequency of this state is the depth estimate of this CRISPR spacer in this sample. Variants detected within the spacer are also contained within the state information. The frequency distribution over the corrected k-mers should be roughly uniform (Figure 11, blue line).

3.4 Results

From sequence data and spacer sequences, this alignment-free spoligotyping approach takes 1 minute and 19 seconds to run on all 92 samples. If Lasergene takes 30 minutes, which is on the lower end of what the *M. tuberculosis* CDC science team was reporting, then an estimated 22x speedup was observed.

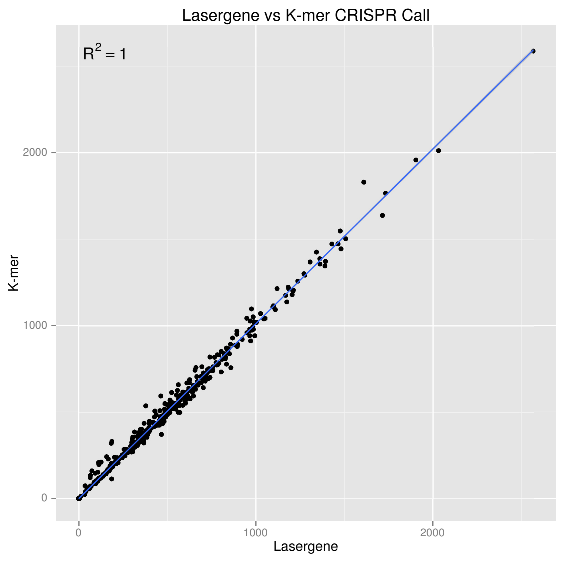

Figure 12: A comparison of spacer depth estimates from k-mer spoligotyping and Lasergene for 92 samples and 68 spacers in each sample.

All 13 SNPs and both deletions were identified and corrected by the software. The depth estimates for each spacer in each sample (6,256 estimates) matched with verified results from Lasergene (Figure 12) with an R^2 of 1.00.

Spacer 16 has a 5x T homopolymer repeat. Pyrosequencing, which is the technology employed by IonTorrent, tends to mis-call the number of bases in these repeat regions [49], and this phenomenon was observed by the software. However, it did not affect the results.

3.5 Discussion

In this limited test case, the alignment-free method of spoligotyping produced perfect results in a fraction of the time when compared to the alignment approach. Although spoligotyping with sequence data was not prohibitive before this software, this does align with my goal of analyzing sequence data using minimum resources.

SpolPred is an existing software application for straintyping *M. tuberculosis* with sequence data [20]. This software matches spacers to sequences, but it only allows a fixed number of varied bases in the form of a hamming distance (distance = 1 by default). It cannot correct for indel mutations, and therefore, it would miss some of the calls in the 92 samples. The alignment-free algorithm presented here is more tolerant to these errors.

Further work is needed to correlate the types of mutations in the spacer regions with their effect on hybridization. This method is potentially more sensitive than the *in vitro* spoligotyping assay because it may detect heavily mutated or partially truncated spacers. However, the nature of these variants and their location in the spacer would be revealed, and a scoring system could be applied to the results to see what the hybridization approach would likely show. I leave this issue for future studies.

What remains to be shown is that the alignment-free method can work on WGS data. To be sure that it could handle the complexity of the whole genome, I thought that the error correction algorithm needed improvements, and so I turned my attention to this issue.

To my knowledge, k-mers have never been used to identify variants from sequence data in a general way, and this is especially true for indels. Some error correction on sequence reads has been done by looking at the k-mer spectrum of a sample and by using hamming distances [119], but there has never been a full-featured variant calling algorithm that uses k-mer frequencies.

This approach is novel and interesting, but it lacks general applicability because of its complexity restraints. Is it possible that the error correction algorithm described here could be applied to broader a variant-calling problem? Could the process be guided so that it was more efficient than $\mathcal{O}(4^n)$? The answer is yes, and CHAPTER IV presents the results of that work.

CHAPTER IV

MAPPING-FREE VARIANT CALLING

4.1 Abstract

Characterizing variants from sequence data is one of the most ubiquitous tasks of bioinformatics because the results are useful for a variety of applications. For example, the etiology of a tumor or its susceptibility to chemotherapeutic drugs can be revealed. Analyzing the variants of bacterial specimens can reveal lineage or virulence characteristics, and by identifying SNPs in many samples, the transmission network of an outbreak can be revealed.

Modern variant calling software relies on sequence read alignments. Although it is by far the *de facto* standard, important variants are missed by the alignment. When SNPs are packed too densely or if there are large indels, the sequence reads in these regions cannot be confidently assigned to the correct location, and so the information contained within them is lost. Variant calling without alignments is one way to solve this problem, but they demand even more computing resources than alignments.

Sequencing platforms are producing more data at a lower cost. As the depth of coverage increases, the confidence of variant calls also increases, but so too does the cost of analysis. Very little research has been dedicated to variant identification without sequence read alignments, but such a method would make analyzing this data less prohibitive.

CHAPTER III introduced a rudimentary variant calling algorithm, but it was too inefficient to scale to large reference sequences. Recognizing that the rebuilding process could be guided by aligning the generated sequence with the reference was the key to making this algorithm tractable. With this modification, the rebuilding

process is streamlined, and the alignment consumes a small fraction of the overall CPU time.

4.2 Introduction

Modern sequencing methods produce a deluge of data. The genome of a sample is often sequenced to a depth of 100 or greater so that for any given base, there are at least 100 sequence reads that represent it. With such depth, the impact of errors in individual sequence is minimized. However, each sequence read covers a tiny fraction of the genome. An Illumina HiSeq instrument is capable of generating sequence reads up to 250 bp long, but when compared to a genome size of 4.4 Mb (*Mycobacterium tuberculosis*), a read only represents 0.0057% of the genome. For the human genome of 3.3 Gb, a 250 bp read represents 0.0000076% of the genome. Even with the best sequence data, finding patterns in a set of these sequence reads is a true technical challenge.

The standard protocol for detecting variation in DNA is to map millions of short sequence reads to a known reference and to find loci where the reference and sequence reads do not agree [96, 92]. This process takes place in two steps; sequence reads are aligned to a reference, and variants are called from the alignment.

This approach works well for most data where variants are not too densely packed and where indels are short. Useful information can be hidden in regions of dense variation and large indels, and so for many applications, it is important to properly characterize these events. In such cases, an alternative to the standard alignment-based variant calling approach is needed.

4.2.1 Alignments

Because it is such a critical task, and because it is technically challenging, sequence alignment has been the subject of intense bioinformatics research, and it has yielded some impressive results. Early alignment tools were designed for the Sanger sequencing method, but as next generation sequencing (NGS) rose, the earlier alignment tools no longer performed well [71]. One category of new aligners arose based on hashing sequence k-mers, and algorithms use a seed-and-extend similar to BLAST [2]. Some hash the sequence reads, such as Eland [7] and MAQ [72]. Eland was developed by Illumina, and MAQ was the first to link the genotype confidence with the alignment confidence. Others hash the reference, such as SOAP [73].

The most advanced sequence aligners to date use a Burrows-Wheeler transform [13] to create an index of the reference sequence that can be efficently searched for inexact matches. An FM-Index [36] compresses the space requirements so that the human genome can be stored in 2-8 GB of RAM [71]. Sequence reads are then compared to this structure and mapped to the genome. This approach is used by Bowtie [64, 63], BWA [68, 69], and SOAP2 [74], and they are some of the fastest and most reliable aligners to date.

4.2.2 Alignment-based variant calling

Once an alignment is obtained, it is searched for differences between the reference sequence and the mapped sequence reads. If a given locus has an "A" in the reference, but the majority of reads show a "G", then there is strong evidence for a A$\rightarrow$G variant at that locus. Short indel variants can also be identified this way.

Given the alignment, a software program must search it for these differences. The naïve approach is to simply scan each base of the alignment for differences, but modern variant callers are more sophisticated. Instead, haplotypes are built from the data, and variants are called from those haplotypes.

The GATK [85] HaplotypeCaller [51] is one of the most heavily used alignment-based variant callers today. This software first identifies *active regions* where the reference and aligned reads do not agree. For each active region, a haplotype is

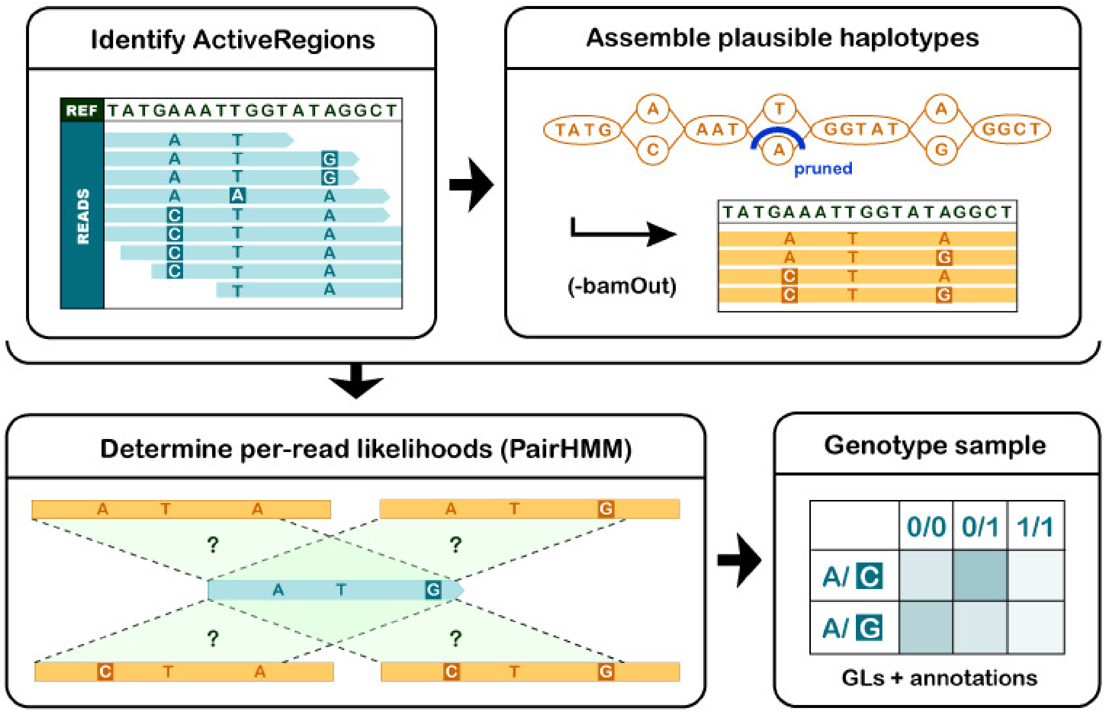

Figure 13: An overview of the GATK HaplotypeCaller. Active regions are identified from the alignment and haplotypes are assembled from sequence reads in the alignment over that region. A hidden markov model aligns the sequence reads to the haplotypes and determines the likelihood of alleles per read for each site with a variant. The likelihood of each variant is then analyzed, and the final variant call is made. Image retrieved from the GATK website [51]

assembled over it using a de Bruijn-like graph structure. Sequence reads aligned to the active region are re-aligned with each haplotype using a *pair hidden Markov model* (PairHMM), and this gives a likelihood for each haplotype based on the sequence read evidence. The likelihood of each variant is determined using Bayes rule and the haplotype likelihood. Figure 13 outlines this process.

FreeBayes [40] is another popular haplotype-based variant caller. Instead of considering the likelihood of a haplotype in one sample, it considers the likelihood from multiple samples. This approach was designed to better characterize variants with different allele frequencies due to copy number alterations.

4.2.3 Limitations of alignment-based variant calling

Although modern alignment tools are designed to handle errors and variation, a sequence read that differs significantly from the reference cannot be confidently assigned to the correct location. When a read is correctly mapped to such a region, it is still difficult to separate true variants from false calls because the variant call confidence is affected by the quality of the alignment of the reads that support it. In extreme cases, the read is clipped or not mapped at all, and there is no alignment coverage at that locus.

There are several alternative approaches to this problem with tools that are already available, but they come with some significant disadvantages. The simplest of these is to perform a *de novo* assembly on the sequence reads to produce scaffolds from the sequence data independent of any reference sequence. These scaffolds can then be treated as very long sequence reads and aligned to the reference. Although dense variation may still occur, the matched bases will likely improve the mapping quality enough to confidently identify variants in those regions.

The assembly alternative assumes that the allele frequency is 1 at all loci, so it can only work for monoploid organisms, and it is incapable of identifying minor alleles that may be present in the sequence data. For example, if a large variant was present in a small frequency, but was selected for when the colony was exposed to an antibiotic, it would be infeasible for this approach to characterize the allele until it reached high-frequency. Furthermore, sequencing depth is essentially collapsed to 1 regardless of the sequence read coverage, and it becomes difficult to correct for false-calls [94]. Also, the de Bruijn graphs used by modern *de novo* assembly software require significant computing resources [75], so this method may take a long time and more expensive equipment to execute.

Cortex [53] was designed to overcome limitations of the *de novo* assembly approach. This software builds a de Bruijn graph from the sequence data and a reference. Instead of collapsing the de Bruijn graph to a single sequence, variants are called directly from it. This preserves important information from the sequence reads, such as read depth and alternate alleles. However, it still requires significant resources to build the de Bruijn graph.

Instead of assembling all sequence reads, Platypus [101] performs local assemblies. This approach is similar to the GATK Haplotypecaller, except that it aligns all reads to all possible haploypes. Therefore, this method incurs the overhead of mapping all reads to the reference if the variant loci are not know *a priori*, and it incurs the overhead of read mapping again on each haplotype.

4.2.4 A new approach

This chapter will outline a novel algorithm capable of characterizing densely-packed SNPs and large indels without mapping, assembly, or de Bruijn graphs. The algorithm and the implementation here are collectively referred to as "Kestrel". Where appropriate, I will make a distinction between the algorithm and the implementation.

Instead of relying on alignments or de Bruijn graphs, the evidence for variant calls comes from k-mer frequencies within the sequence data. Like the other methods described here, active regions are identified and haplotypes are assembled over them. However, Kestrel uses k-mer frequencies as evidence to support both steps. A novel alignment algorithm guides the haplotype reconstruction process, and this greatly reduces the amount of time and resources spent resolving alignments. The result is an impressively fast variant caller that is capable of characterizing regions with dense variation and large indels.

This approach is the first in its class. In my research, I have not seen anything like this algorithm in the literature or reduced to practice.

4.3 Methods and algorithms

The Kestrel algorithm identifies active regions and assembles haplotypes over those regions. The active region identification is performed using a sequence of expected k-mers from the reference and the k-mer frequencies from the sequence reads. Once an active region is identified, one or more haplotypes are assembled over it using a novel alignment algorithm to guide the process. Variants are easily identified from the alignment of the active region with the haplotype. The following sections outline this process method in detail.

4.3.1 Kestrel Overview

Using KAnalyze [4], the frequency of each k-mer in the sample is stored in an indexed k-mer count file (IKC) (Figure 14(a)). The IKC structure is a minimizer-grouped set of k-mers outlined in Section 2.3.6. From the reference sequence, the frequencies for each k-mer and its reverse complement are obtained from the IKC file and summed, but these are left in order (Figure 14(b)). Kestrel searches the resulting array for loci where the frequency declines and recovers (Figure 14(c)), which suggests the k-mers of the reference and sample differ. Analogous to the GATK[85] HaplotypeCaller, this low-frequency region is called an *active region*, and *haplotypes* are reconstructed over it. The high frequency k-mers at the ends, the *anchor k-mers*, are part of this region, and they seed the haplotype reconstruction process.

Starting with the left anchor k-mer, the first base is removed, all four bases are appended to this (k - 1)-mer, and the k-mer frequency is queried for each of the four possibilities (Figure 14(d)). An equivalent process is performed on the reverse complement k-mer, and the frequencies are summed. The base that yields a high frequency k-mer is appended to the haplotype. The new k-mer ending in that base is then used to find the next base by the same process. If more than one of these k-mers has a high frequency, a haplotype is assembled for each one.

A modified Smith-Waterman [111] algorithm guides the process by aligning the active region and the haplotype as it is reconstructed (Figure 14(e)). By setting an initial score and disallowing links to zero score states, alignments are anchored on the left and allowed to extend until an optimal alignment is obtained. Variant calls follow trivially from the alignment (Figure 14(f) and Figure 14(g)).

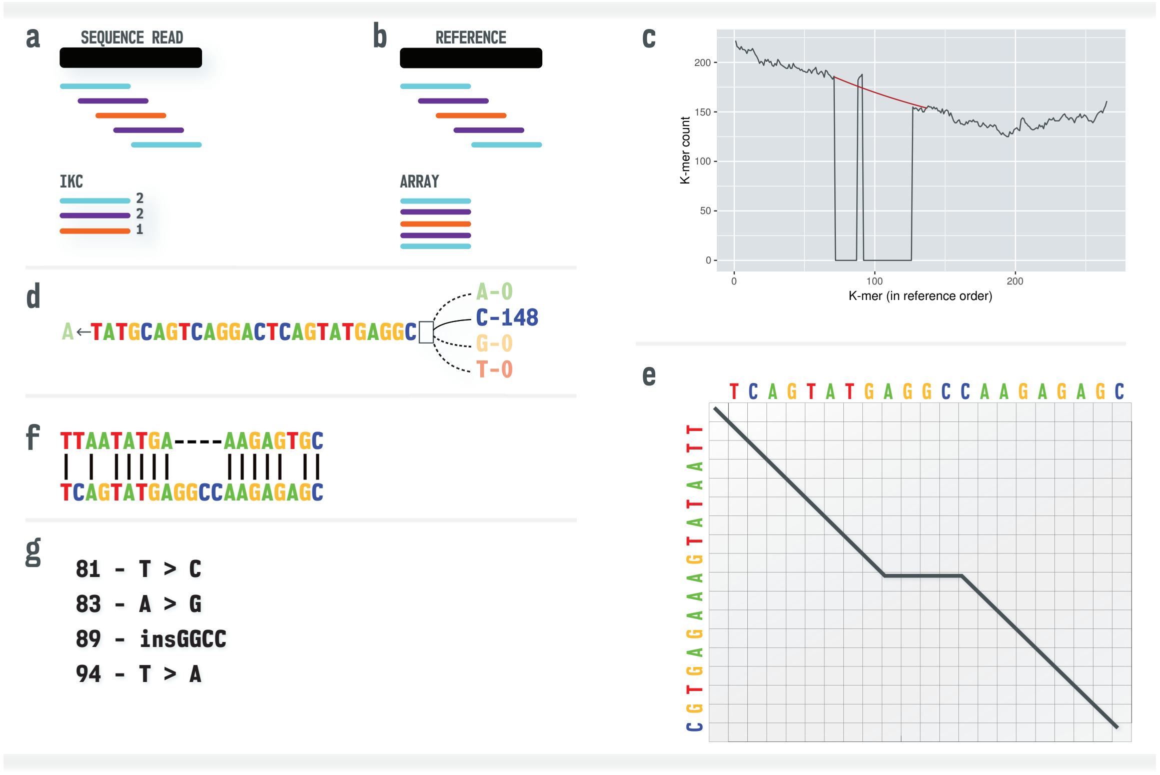

Figure 14: Overview of the Kestrel process from sequence data to variant call. (**a**) The sequence reads are converted to an IKC file. (**b**) The reference sequence is converted into an array of k-mers. (**c**) K-mer frequencies from the sequence reads (vertical axis) are assigned to the k-mers of the reference (horizontal axis). A decline and recovery of the frequencies bound an active region where one or more variants are present. (**d**) Starting from the left anchor k-mer (last k-mer with a high frequency), the first base is removed, each possible base is appended, and the base that recovers the k-mer frequency is appended to the haplotype. (**e**) A modified alignment algorithm tracks haplotype reconstruction and terminates the process when an optimal alignment is reached. (**f**) This algorithm yields an alignment of the reference sequence and haplotype within the active region. (**f**) Variant calls are extracted from the alignment. Image created by Dawn Audano.

This section gives a brief overview of the process to give the reader context for what follows. The details of each step are outlined below.

4.3.2 Preliminaries

Given a known reference sequence, X, identify active regions, x, which are suspected to contain variants. Using evidence from the sequence reads, dynamically find one or more haplotypes, y, that align with x. x and y must align end-to-end, and may contain mismatched bases or gaps in either sequence. Building y and calling variants from it is the aim of the Kestrel algorithm.

X	:=	Full reference string.
x	:=	A substring of X, called the active region, where variant detection occurs.
y	:=	A haplotype over x, which is discovered by the Kestrel algorithm.
k	:=	The number of bases in each k-mer.

4.3.3 Locating active regions

N is an array of frequencies for each k-mer in the reference sequence, X, from the first k-mer (N_1) on the left end to the last k-mer on the right end of X. $N_i = 0$ if the k-mer at position i contains ambiguous bases. If X contains no variants, and therefore no active regions, the distribution of the frequencies in N is roughly uniform with some fluctuation due to sequence read errors, non-uniform read coverage, k-mer overlap with other regions of the genome, and other sequencing anomalies.

N is traversed from left to right searching for a sharp decline or increase of the frequency between neighboring k-mers N_i and N_{i+1}, which may indicate the edge of an active region. The threshold, ϵ, is the magnitude of difference between N_i and N_{i+1} that must be exceeded to trigger an *active region scan*.

If $N_i > N_{i+1}+\epsilon$, then the active regions spans from N_i to some downstream k-mer, $i + l$, where the k-mer frequency returns to a value near N_i. The active region, x, defined by this range includes both N_i and N_{i+l}, which are the anchor k-mers, and they help seed and terminate the process of building y. If y does not start and end with these anchor k-mers, then y is rejected.

The shortest active region is the case when one or more bases are inserted and no other variants occur within k bases of the insertion. In this case, the number of k-mers with a frequency affected by this insertion will be $k - 1$. Since an active region, x, includes one unaltered k-mer on each end (anchor k-mers), x must span $k + 1$ k-mers or it is rejected.

If a variant occurs less than k bases from the right end of the sequence, X, then there will be no anchor k-mer on the right side because all k-mers up to the end of X are altered. In this case, x may be left un-anchored on its right end, and y is not required to match the missing anchor k-mer. y must also be allowed to end with a deletion from x. Since the evidence for discovered variants is not as strong, Kestrel requires both anchor k-mers by default.

If $N_i < N_{i+1} - \epsilon$, then the k-mer frequencies increased significantly, and an active region may be present from N_1 to N_{i+1}. Similar to an active regions scan that reaches the right end of N, this will occur if variants are found less than k bases from the left end of X. This region is anchored on the right side by N_{i+1}, but it is not anchored on the left side. In this case, y may begin with a deletion on the left side. Since this case requires that active regions are built from right to left, Kestrel reverses the sequence of the active region, builds in the left to right direction, and reverses the result. Note that sequences are only reversed and not complemented. As with the case where the right anchor k-mer is missing, this active region will be ignored by default because the evidence for variants will not be as strong.

Real data is not uniform, so several heuristics are employed to aid active region detection. Because the read coverage may decline, an exponential decay function is applied to the recovery threshold. K-mers may map to more than one region, and this would create peaks in the sequence data, and these peaks must be detected and ignored. Section 4.3.13 discusses these active region detection heuristics in more detail.

ϵ is chosen automatically by Kestrel based on the distribution of reference k-mer frequencies, and this process is discussed in Section 4.3.11.2.

N	:=	A set of k-mer frequencies ordered by the k-mers in X.
ϵ	:=	K-mer frequency difference threshold.
l	:=	The number of k-mers from the first low-frequency k-mer to the right anchor k-mer.

4.3.4 Haplotype reconstruction

Starting with the left anchor k-mer, the haplotype is reconstructed from left to right. The active region contains the anchor k-mer, and so this k-mer is first added to the alignment. The first base is removed from the k-mer, and each of the possible four bases are appended to the resulting $(k-1)$-mer. The base that produces the highest count is added to the alignment, and the new k-mer ending in that base seeds the next step.

Algorithm 4 shows a summary of this reconstruction process. Base A is added to the $(k-1)$-mer first, and the count of that k-mer is highest by default. C is then added to the $(k-1)$-mer, and if it produces a higher k-mer count, it replaces A as the base to be added, and the maximum count, $count_{max}$ is updated. The same process is repeated for G and T. The base that produced the highest count in the end is appended to the alignment.

For illustration purposes, Algorithm 4 is simplified in two ways. First, it does not show how the reverse-complement k-mer is used to include information from the other DNA strand. When the $(k-1)$-mer is created, the same is done for the reverse-complement, except the last base is removed instead of the first. When a base is

Algorithm 4: Alignment-guided haplotype reconstruction process.

```
Input: $kmer$: The anchor k-mer that seeds haplotype reconstruction.
for each base in kmer do
    addBase(base)                                ▷ Add the anchor k-mer to the alignment
repeat
    $kmer_{sub} \leftarrow$ shift(kmer, 2)       ▷ Create the (k - 1)-mer

    $kmer_{test} \leftarrow$ append($kmer_{sub}$, A)     ▷ Append A
    $count_{max} \leftarrow$ getCount($kmer_{test}$)
    $base \leftarrow A$

    $kmer_{test} \leftarrow$ append($kmer_{sub}$, C)     ▷ Append C
    $count \leftarrow$ getCount($kmer_{test}$)
    if $count > count_{max}$ then
        $count_{max} \leftarrow count$
        $base \leftarrow C$

    $kmer_{test} \leftarrow$ append($kmer_{sub}$, G)     ▷ Append G
    $count \leftarrow$ getCount($kmer_{test}$)
    if $count > count_{max}$ then
        $count_{max} \leftarrow count$
        $base \leftarrow G$

    $kmer_{test} \leftarrow$ append($kmer_{sub}$, T)     ▷ Append T
    $count \leftarrow$ getCount($kmer_{test}$)
    if $count > count_{max}$ then
        $count_{max} \leftarrow count$
        $base \leftarrow T$

    if $count_{max} > 0$ then
        $kmer \leftarrow$ append(kmer, base)     ▷ Update k-mer
        addBase(base)                            ▷ Append to alignment
until Alignment cannot improve with more bases or $count_{max} == 0$
```

appended to the $(k-1)$-mer, its complement is prepended to the reverse-complement of the $(k-1)$-mer, and the counts are summed. Second, it does not illustrate how multiple haplotypes are allowed within the active region. Whenever the maximum count is greater than 0 and a base is appended that produces a k-mer count greater than 0, they are compared. The lesser of the two is saved to a state that is later resumed, and the reconstruction process continues with the base that produced the higher count. This way, any number of haplotypes can be discovered for the active region.

In noisy data, an excessive number of haplotypes can be discovered. Therefore, Kestrel limits the number of states that can be saved. It is for this reason that the algorithm checks the highest k-mer count before deciding which one to save and which one to continue building. The haplotype states are alse saved in a way so that the least likely one is dropped when the cache limit is exceeded.

The following sections describe the alignment that guides reconstruction.

4.3.5 Alignment parameters

The alignment step is a dynamic programming algorithm based on the well known Smith-Waterman [111] algorithm. The key difference is that only the active region sequence, x, is known *a priori*, and each haplotype, y, is built using a local assembly approach that terminates when an optimal alignment obtained. All acceptable alignments must match each base of the left anchor k-mer in x and y, but the right end of y is not known. Since this alignment must guide the process of building y, it must anchor on one side, but extend arbitrarily as y is reconstructed.

Two key modifications were made to Smith-Waterman; (i) any subalignment with a score of 0 cannot be extended, and (ii) the alignment must begin with a score greater than 0. These modifications force all possible alignments terminate on the left side as if it were a global alignment, but update dynamically as y is extended. Section 4.3.10 outlines how the scores are used to determine when to stop building y.

As with other alignment algorithms, a set of scoring criteria is required, and the alignment is optimized over these parameters. When two bases are aligned, R_{match} is added to the alignment score if the bases match, and $R_{mismatch}$ is added when the bases do not match. For each base aligned with a gap, R_{gap} is added to the score. $R_{gapopen}$ is added to the score for each time a gap is opened, i.e., once for every maximal substring consisting of gap character, “-”. The initial score of the alignment is R_{init}, as required by our modification to Smith-Waterman. Note that R_{match} and R_{init} are strictly positive, $R_{mismatch}$, and R_{gap} are strictly negative, and $R_{gapopen}$ is non-positive. The optimal score matrix will be determined by the application, such as how divergent the sequences may be and the size of allowable insertion or deletion events.

$$\begin{aligned}
R_{match} &\coloneqq \text{Aligned bases match, } R_{match} > 0 \\
R_{mismatch} &\coloneqq \text{Aligned bases do not match, } R_{mismatch} < 0 \\
R_{gapopen} &\coloneqq \text{A gap was opened, } R_{gapopen} \leq 0 \\
R_{gap} &\coloneqq \text{Base aligned with a gap, } R_{gap} < 0 \\
R_{init} &\coloneqq \text{Initial alignment score, } R_{init} > 0
\end{aligned}$$

For convenience, we define a function, $match(i, j)$, to return the appropriate score for aligned bases, as shown by Equation 2.

$$match(i,j) = \begin{cases} R_{match} & : x_i = y_j \\ R_{mismatch} & : x_i \neq y_j \end{cases} \tag{2}$$

4.3.6 Alignment data structures

The optimal alignment has the highest score of all possible alignments of x and y. Trying all possible alignments is clearly an inefficient way of solving the problem, however, the score of one particular alignment can be seen as the score of the same alignment one position shorter plus the score of the last position [27]. The dynamic programming solution effectively *memoizes*[1] the score of shorter alignments in a score matrix instead of recomputing it.

[1]“Memoization” is a computer-science technique for caching and reusing intermediate results.

The alignment data structures are analogous to those typically employed in Smith-Waterman implementations using an affine gap model. One score matrix, S_{aln}, tracks scores through aligned (matched or mismatched) bases. Two more score matrices, S_{gact} and S_{ghap}, track alignment scores through gaps in the active region (x) or a gaps on the haplotype (y), respectively. The bases of x are positioned along the vertical axis of each matrix, and the bases of y are positioned along the horizontal axis.

The first base of x and y are represented in row 1 and column 1, respectively. Each base of the anchor k-mer is added to y, and one column is created in all matrices for each base of y. The initial alignment score, R_{init} is assigned over the anchor k-mer (Equation 3), and all other scores in all three matrices are initialized to 0. A fourth matrix, T, contains traceback information from the end of an alignment to $S_{aln}(0, 0)$. It is initialized so that there is one sub-alignment over the anchor k-mer (Equation 4). This initialization of S_{aln} and T matches each base of the active region and haplotype over the anchor k-mer that seeds haplotype reconstruction, and all acceptable alignments must enter this path at $S_{aln}(k, k)$.

$$S_{aln}(i, i) = R_{init} \; \forall i \in \mathbb{Z}, 0 \leq i \leq k \tag{3}$$

$$T(i, j) \rightarrow T(i-1, j-1) \; \forall i \in \mathbb{Z}, 1 \leq i \leq k \tag{4}$$

Storing the whole of all four matrices would result in a large memory footprint, and Section 4.3.12 outlines how the software implements this algorithm more efficiently.

S_{aln}	:=	Score matrix for alignments through aligned bases.
S_{gact}	:=	Score matrix for alignments through insertions (active region gaps).
S_{ghap}	:=	Score matrix for alignments through deletions (haplotype gaps).
T	:=	The traceback matrix.

4.3.7 Alignment score matrices

The score matrices are built using the usual alignment algorithm, but with the Kestrel modifications. For example, transitioning to $S_{aln}(i,j)$ requires adding $match(i,j)$ to each $S_{aln}(i-1,j-1)$, $S_{gact}(i-1,j-1)$, or $S_{ghap}(i-1,j-1)$ that are above 0 and choosing the maximum value. $T(i,j)$ is updated to link $S_{aln}(i,j)$ to the cell or cells that yielded the maximum score. If all scores in $S_{aln}(i-1,j-1)$, $S_{gact}(i-1,j-1)$, and $S_{ghap}(i-1,j-1)$ are 0, or if the computed score is 0 or less, then $S_{aln}(i,j) = 0$ and no link is added to T. Equation 5 outlines assignment of $S_{aln}(i,j)$.

$S_{gact}(i,j)$ is set by finding all non-zero scores from $(i,j-1)$ in each score matrix. $R_{gapopen}$ is added to the scores from S_{aln} and S_{ghap}, and R_{gap} is added to all scores. If it is above 0, then the maximum score is recorded in $S_{gact}(i,j)$ and T is updated. Equation 6 describes this calculation, and Equation 7 describes a similar calculation for S_{ghap}.

$$S_{aln}(i,j) = \max\begin{cases} 0 & \\ S_{aln}(i-1,j-1) + match(i,j) & : S_{aln}(i-1,j-1) > 0 \\ S_{gact}(i-1,j-1) + match(i,j) & : S_{gact}(i-1,j-1) > 0 \\ S_{ghap}(i-1,j-1) + match(i,j) & : S_{ghap}(i-1,j-1) > 0 \end{cases} \quad (5)$$

$$S_{gact}(i,j) = \max\begin{cases} 0 & \\ S_{aln}(i,j-1) + R_{gapopen} + R_{gap} & : S_{aln}(i,j-1) > 0 \\ S_{gact}(i,j-1) + R_{gap} & : S_{gact}(i,j-1) > 0 \\ S_{ghap}(i,j-1) + R_{gapopen} + R_{gap} & : S_{ghap}(i,j-1) > 0 \end{cases} \quad (6)$$

$$S_{ghap}(i,j) = \max\begin{cases} 0 & \\ S_{aln}(i-1,j) + R_{gapopen} + R_{gap} & : S_{aln}(i-1,j) > 0 \\ S_{gact}(i-1,j) + R_{gapopen} + R_{gap} & : S_{gact}(i-1,j) > 0 \\ S_{ghap}(i-1,j) + R_{gap} & : S_{ghap}(i-1,j) > 0 \end{cases} \quad (7)$$

4.3.8 Alignment extension

Recall from Section 4.3.3 that the start of the active region, x, was found by locating neighboring k-mers where the frequency difference exceeded a threshold, $N_i > N_{i+1} + \epsilon$. The haplotype, y, is initialized using the k-mer associated with N_i, the left anchor, and as described in Section 4.3.6, all bases in that anchor initialize the alignment.

The next k-mer after the anchor is altered, and so it has a low frequency. Since the anchor k-mer shares $(k-1)$ bases with the next k-mer, it can be permuted to find what the next base should be. This starts with removing the first base of the anchor to create a $(k-1)$-mer. Then, each possible base is appended to this $(k-1)$-mer, and frequency for each resulting k-mer is retrieved. The base with the maximum frequency is appended to y, and the alignment is updated. The new k-mer is then used to find the next base, and the process repeats until y is fully constructed. Because active region detection takes place using the forward and reverse-complement k-mers, a parallel process on the reverse-complement is performed, and the k-mer frequencies are summed.

If no bases produce a k-mer with an acceptable frequency, the alignment is terminated. If more than one base produces an acceptable k-mer frequency, then the alignment state is saved and resumed after another alignment completes. Therefore, more than one haplotype, y, may be found for each active region x.

If the active region has no left anchor, and if variant calling without both anchors is enabled, then its right anchor seeds the alignment. In this case, whole alignment process takes place in reverse. Each y is then reversed back to its original configuration to match x after this step completes.

4.3.9 Maximum score and optimal alignments

After each base of y is added to the alignment, the overall score can be examined. Since the alignment must cover all of x, only the last row of the alignment score matrix, S_{aln}, needs to be queried. Equation 8 defines R_{max}, the maximum alignment score.

$$R_{max} = \max\left(\{S_{aln}(|x|, j)\ \forall j,\ 0 \leq j \leq |y|\}\right) \tag{8}$$

If x extends to the end of the reference, X, then it is possible that the alignment ends in a deletion because there is no anchor k-mer on that end. In this case, the maximum score can be computed as the maximum of the final row in both S_{aln} and S_{ghap}. The maximum score is never calculated from the S_{gact} because inserting bases on the end of x can only lower the maximum score. The full alignment is found by traversing the traceback matrix, T, from the cell with the maximum score to $S_{aln}(0, 0)$.

If any cell of T has more than one path out, then there is more than one optimal alignment, and the first alignment as defined by the alignment sort order is used. When comparing two alignments, the one with the first non-matching base comes first. If the non-matching bases agree (same variant), then the next non-matching base is queried. If the non-matching bases do not agree, then alignments are prioritized by mismatch, insertion, and deletion, in that order. This gives the algorithm predictable output for cases such as a deletion in a homopolymer repeat; Kestrel will always report that the first base was deleted even though the alignment score would be the same for a deletion at any locus of the repeat.

4.3.10 Alignment termination

The extension of y must be terminated when the best possible score is reached. Each time a base is added to the alignment, the maximum alignment score is known. However, the maximum possible alignment score that could be obtained by adding more bases to y must also be known. When this maximum potential score is less than the maximum alignment score, then the alignment cannot be improved by extending

y further.

The maximum potential score is determined by examining the last column of S_{aln}. The best possible score that can be obtained from any cell is the case where all subsequent bases of x are aligned with matching bases in y. Equation 9 defines $maxpot(i, j)$, the maximum potential score from cell $S_{aln}(i, j)$. Equation 10 defines R_{maxpot} as the maximum $maxpot(i, j)$ of the last column of S_{aln}. If $R_{max} > R_{maxpot}$ (Equation 8), a greater or equal score cannot be obtained by adding more bases to y.

$$\begin{aligned} maxpot(i,j) &= \begin{cases} S_{aln}(i,j) + (|x| - i) \cdot R_{match} & : S_{aln}(i,j) > 0 \\ 0 & : S_{aln}(i,j) = 0 \end{cases} && (9) \\ R_{maxpot} &= \max\left(\{maxpot(i, |y|),\ 0 \leq i \leq |x|\}\right) && (10) \end{aligned}$$

When the alignment extends to an end of the reference sequence, then the maximum potential score from the deletion score matrix, S_{ghap}, must also be considered.

The modifications to Smith-Waterman outlined in Section 4.3.5 are important to make the termination condition deterministic when y does not align well with x. Firstly, the algorithm gives up on alignments that pass through a cell with a zero score. This is also the reason R_{gap} may not be 0; if it were 0, then the alignment could attach to another region of the genome and extend without bound. These modifications allow degenerate cases to be limited before spending many CPU cycles trying to solve it.

4.3.11 Parameter selection

4.3.11.1 K-mer size

The most visible parameter is the k-mer size, k, which should be selected to balance genome complexity with expected error rates. The majority of k-mers must match one region of the genome. When a k-mer maps to multiple regions, the frequencies from both will be mixed together. This will hinder both active region detection and assembly. If the k-mer size approaches the read size, few k-mers will be extracted from each read and the observed coverage will decline. A high error rate in the sequencing data will also cause an observed loss of coverage. Section 2.2.3 contains more information about this topic.

In my research, I have not encountered many reliable rules for choosing the k-mer size for this type of problem. I have also not dedicated much time to it because I have not found it difficult to pick sizes that worked for my purposes. This parameter appears to place rigid bounds on the performance of k-mer algorithms, but I believe it is far more flexible than it appears. Unless the error rate is very high, there tends to be a flexible range for the k-mer size where this algorithm works well. In my experience, I have used 31-mers for bacterial species, and 48-mers for human data. Note that some repeat regions are longer than any k-mer or sequence read, and these will still cause issues regardless of the chosen size.

Clearly, more research using diverse data sets could be dedicated to providing a more complete answer to this question.

4.3.11.2 Difference Threshold

Active regions are detected when the absolute difference between neighboring k-mers exceeds some threshold, ϵ. This parameter is selected by choosing a quantile, Q_ϵ, over the absolute differences of all neighboring k-mer frequencies. Of all frequency differences $|N_i - N_{i+1}|$, Q_ϵ of these will be large enough to trigger an active region scan. Choosing this parameter depends on how many active regions can be expected, although, a value as high as .95 to .99 works in many cases.

$$Q_\epsilon \quad := \quad \text{A quantile of the frequencies of } N \text{ for choosing } \epsilon.$$

4.3.12 Alignment matrix implementation

Storing the whole of all four alignment matrices defined in Section 4.3.6 would use an excessive amount of memory ($\mathcal{O}(n^2)$). This problem would be compounded when multiple haplotypes are searched because all matrices would need to be duplicated. Fortunately, only a small fraction of this data needs to be stored, and the memory requirements can be reduced significantly ($\mathcal{O}(n)$).

Because of the nature of the dynamic programming algorithm, only the last column of the score matrices (S_{aln}, S_{gact}, and S_{ghap}) needs to be stored while the next column is built. Therefore, each of these matrices can be reduced to two arrays where one contains the last column, and one contains the new column being added. When another haplotype is found, the alignment splits. An alignment with one base will continue, but the other needs to be saved and restored later. When this occurs, only one array for each matrix needs to be stored.

The traceback matrix, T, is more complex because it is not stored as a matrix. Instead, it is a linked-list of alignment states. Following the links always leads back to $S_{aln}(0,0)$, which is where all alignments begin. When a non-zero score is added to a score matrix, a link is added to T. This slightly complicates the score matrices because they must store a score, and for all non-zero scores, they must also store a node in T.

The nodes in T must also contain more than one link in the case that there is more than one optimal alignment path. Where the first link traverses back toward $S_{aln}(0,0)$, the second link points to other nodes at the same level. When this second link is not `NULL`, the alignment splits into multiple paths at that location.

Since the active region must be aligned from end to end, the only acceptable alignments contain a non-zero score in the bottom row of the matrix. Since the whole score matrix is not saved, the optimal alignment score must be tracked for each new column, and the optimal node is saved separately from the score matrices. The score and link into T is stored. When a column is added that is greater, it's link and score replaces the previous one. As with the linked list, more than one node may have the high score, and so a list of nodes into T are stored for this case.

In the case that multiple haplotypes are found, the alignment must split. As already noted, only the last column of each score matrix must be stored, and this column contains links into T. Since T is a linked list that is only traversed toward $S_{aln}(0, 0)$, one node of the alignment may have several links into it. Therefore, different haplotypes may link to the same node in T where it split and no part of T needs to be duplicated or saved other than the nodes already stored in the score matrix array. Both haplotypes will trace back to the point where they merged and continue on toward $S_{aln}(0, 0)$.

The linked list structure has another more subtle property that Java uses to keep memory usage low. When an alignment path reaches a dead end, the node at the end of the path has no reference to it. This allows the Java Virtual Machine (JVM) garbage collector to detect and remove these nodes. In other words, dead branches of T are automatically pruned. This improves scalabilty by reducing the memory requirements for large active regions where many haplotypes are investigated.

4.3.13 Active region heuristics

In an ideal scenario, sequence data would cover the sample uniformly at all loci, contain no error, and k-mers would be long enough so that no k-mer would map to any other region of the genome. This is never observed in practice. Sequence data does contain errors, and it is almost never distributed uniformly. Furthermore, it may be impossible to choose a k-mer size that eliminates all clashes with other loci. For an algorithm that relies on k-mer frequencies, this presents a challenge. These situations must be carefully handled by additional heuristics to avoid errors in the results.

4.3.13.1 Exponential decay

While scanning for the end of an active region, the active region detector searches for a k-mer frequency that is close to the frequency of the anchor k-mer. When the read depth is not uniform over the active region, this recovery threshold may never be found, and the scan may reach the end of the reference sequence.

To address this problem, an exponential decay function, $f(x)$, is employed to reduce the recovery threshold as the active region extends. $f(0)$ is the anchor k-mer frequency, and it approaches a lower bound, f_{min}, asymptotically. By default, $f_{min} = 0.55 \cdot f(0)$ to avoid ending an active region prematurely on a heterozygous variant. $f(x)$ is defined by scaling and shifting the standard exponential decay function, $h(x)$, as highlighted by Equation 11 and Equation 12.

$$h(x) = e^{-x\lambda} \tag{11}$$

$$f(x) = (f(0) - f_{min}) \cdot h(x) + f_{min} \tag{12}$$

$h(x)$ is parameterized by λ, which must be also be set, but Kestrel does not configure this parameter directly because it is difficult to know how to choose a reasonable value. Instead, λ is chosen by a configurable parameter, α, that is defined as the proportion of the decay range, $f(0) - f_{min}$, after k k-mers. This provides a more intuitive way to define how rapidly the recovery threshold is allowed to decline. Choosing λ given α is shown in Equation 13. Figure 15 illustrates exponential decay with two values of α.

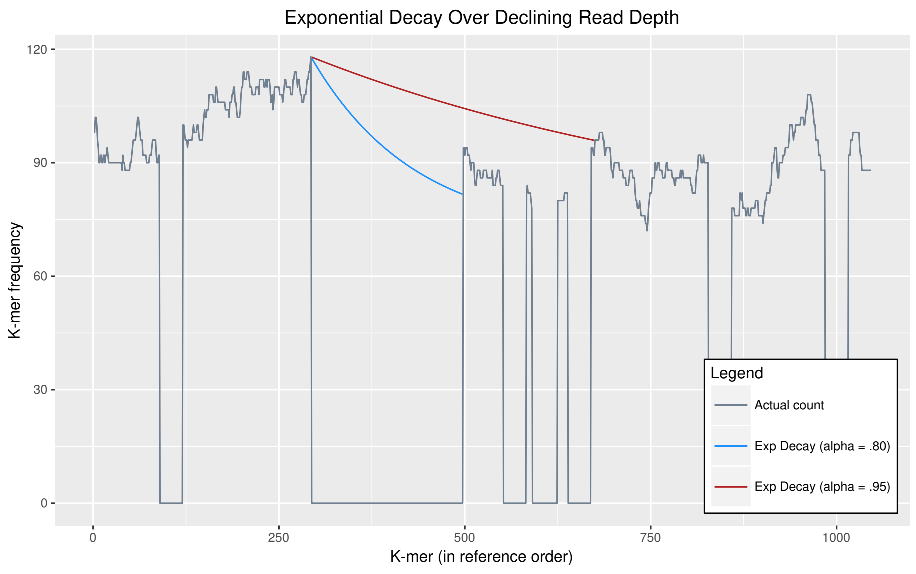

Figure 15: To end an active region, the Kestrel algorithm searches for a k-mer frequency that is close to the frequency anchor k-mer. An exponential decay function is applied to the recovery threshold so that the value declines asymptotically as the scan moves to the right.

$$\begin{aligned} h(k) &= \alpha \\ e^{-k\lambda} &= \alpha \\ -k\lambda &= log(\alpha) \\ \lambda &= \frac{-\log(\alpha)}{k} \end{aligned} \tag{13}$$

At k k-mers, the recovery threshold $f(k) = \alpha \cdot (f(0) - f_{min}) + f_{min}$. In other words, $f(k)$ has declined in its range from $f(0)$ to f_{min} by a factor of α. This is true for all nk such that $f(nk) = \alpha^n \cdot (f(0) - f_{min}) + f_{min}$. Figure 16 gives a proof for this claim.

$f(x)$:= Exponential decay function for the active region recovery threshold.

$h(x)$:= The standard exponential decay function, $e^{-x\lambda}$.

f_{min} := Lower bound of $f(x)$.

$$
\begin{aligned}
h(x) &= e^{-x\lambda} \\
&= e^{-x\frac{-\log(\alpha)}{k}} \\
&= \left(e^{\log(\alpha)}\right)^{\frac{x}{k}} \\
&= \alpha^{\frac{x}{k}}
\end{aligned}
$$

$\square$

Figure 16: A proof of the claim that $f(k)$ declines by $\alpha^{\frac{x}{k}}$. For every $x = nk$, the decay function has declined α^n.

4.3.13.2 Peak detection

Due to chance or homology, peaks may be observed in the k-mer frequencies of a reference, N. If k is not large enough so that all k-mers map to exactly one locus, then the frequency of some k-mers will be higher because it was found in multiple loci. This causes a peak in the data, and if not properly handled, can lead to an arbitrary active region scan or the premature end of an active region. Figure 17 shows a peak in an active region where some reference k-mers happened to match another locus. If the active region scan ended on that peak, variants would be missed.

To deal with this problem, Kestrel does not stop immediately when it finds a peak. Instead, it scans ahead to see if the frequency declines again after some number of k-mers (7 k-mers by default). If it does decline within that threshold, then Kestrel passes the peak and continues as if it were not present.

Occasionally, a scan through an active region will encounter many peaks because the scan recovery threshold is at the level of the k-mer frequencies in that region. These peaks are normal fluctuations in the frequencies. When Kestrel finds many peaks, it goes back to the last sharp increase of k-mer counts within the active region scan. If there was no sharp incline, the scan is abandoned and no active region is declared.

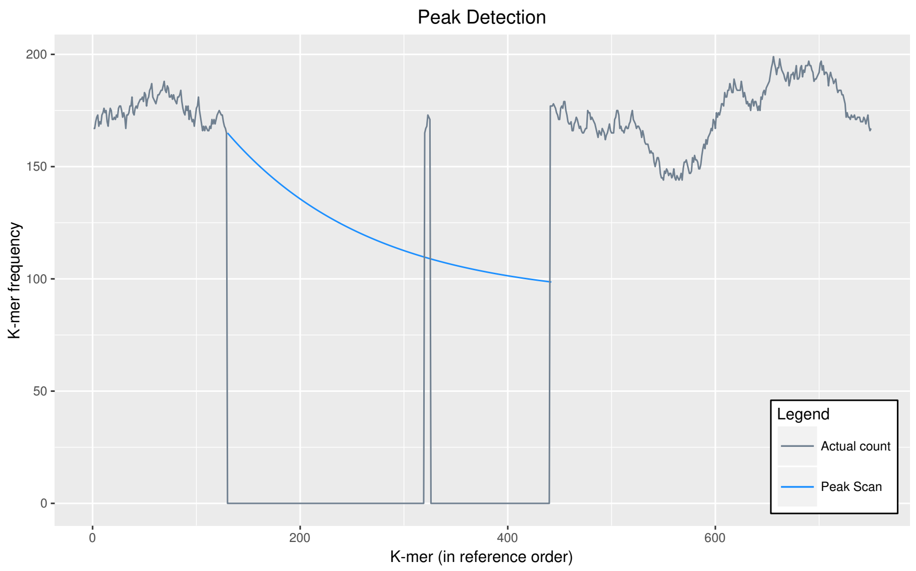

Figure 17: Peaks must be detected and bypassed to prevent prematurely ending an active region scan.

4.4 Results

4.4.1 Motivation

Streptococcus pneumoniae (*S. pneumoniae*) is known to cause severe respiratory and systemic disease including pneumonia, meningitis, and sepsis. Mortality and morbidity is particularly high in immunocompromised individuals [34], and it is responsible for up to 11% of child deaths [93].

β-lactam compounds constitute a large family of broad-spectrum antimicrobial penicillin-like drugs. Their structural similarity to a component of pepditoglycan allows them to bind and covalently link to a serine in the active site of proteins involved in cell wall synthesis. The *S. pneumoniae* penicillin binding protein (PBP) genes are the targets of these drugs.

Many bacterial species evade these drugs by expressing β-lactamases capable of degrading the antibiotic compound, however, these genes are rarely found in *S. pneumoniae*. Instead, the mechanism of *S. pneumoniae* resistance is its ability to significantly alter its PBP genes with homologous extracellular DNA [46]. In several studies, 20% or more of a PBP gene may be altered by inter-species recombination [83, 61], and this can create mosaic PBP genes with a lower β-lactam binding affinity. Because these recombination events may alter hundreds of contiguous bases, variant calling from a standard alignment pipeline cannot characterize them.

4.4.2 Data

Testing was done with 181 whole genome sequence (WGS) samples representing 29 serotypes that were recently released by the Centers for Disease Control and Prevention in NCBI under BioProject PRJNA284954. These data are whole-genome 250 bp paired-end Illumina sequence reads ranging from 14 Mbp to 1,176 Mbp (median = 308 Mbp).

All variants were called against a single reference, TIGR4 (NC_003028.3). Selecting the best *S. pneumoniae* reference out of more than 10 is often necessary [67] for the alignment approach, however, Kestrel was tested to see how well it generalize when the lineage of the reference sequence diverged from the sample. Figure 18 shows the relationship of all samples as a phylogenetic tree based on *average nucleotide identity* (ANI) [44].

Contamination was identified in samples SRR2072298, SRR2072306, SRR2072339, SRR2072342, SRR2072351, and SRR2072379. Each of these exhibited many variant calls in the PBP genes with a relative depth of 0.70 or less. The size of the IKC files is also larger than expected, which supports my conclusion that there is sequence data in these FASTQ that does not belong the sample. Based on the IKC files, two more samples (SRR2072345 and SRR2072352) are also likely contaminated, but since I saw no evidence for this in the variant calls over the PBP genes, these samples were included in the analysis.

Two other samples were removed for having incomplete sequence data: SRR2072219 and SRR2072360. They contain 10 Mb and 37 Mb, respectively, where the median has 323 Mb and the next lowest sample has 92 Mb. Figure 21(d) shows the contaminated sample floating to the top of the IKC file size distribution, and the low-coverage samples at the bottom left.

After removal, 173 samples remained for testing.

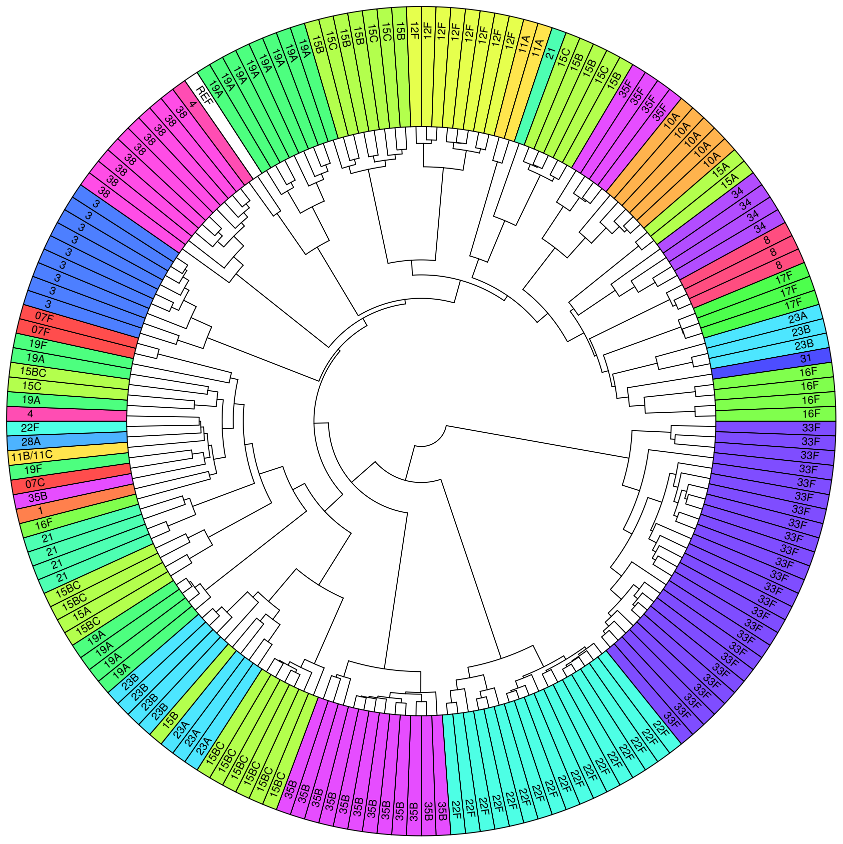

Figure 18: Phylogeny by serotype. The reference is colored white, and each major serotype group is assigned a unique color.

4.4.3 Methodology

Variants were detected in four PBP genes (PBP2X, PBP1A, PBP2B, and PBP2A) with three distinct approaches. The first is a standard alignment pipeline using BWA [68, 69], Picard [52], and GATK [85] HaplotypeCaller. The second is Kestrel. The third is a *de novo* assembly pipeline using SPAdes [5], BWA, and SAMtools [70].

4.4.3.1 Assembly

For the purposes of this experiment, the alignment and Kestrel approaches are tested against the assembly approach. Variants identified by the assembly where the depth of aligned scaffolds is exactly 1 were defined as the true variants for verifying HaplotypeCaller and Kestrel variant calls.

All sequence reads for each sample were assembled with SPAdes using default options. The scaffolds were aligned to the TIGR4 reference using BWA mem and setting the mismatch penalty to 1 (-B 1) and the clipping penalty to 1,000 (-L 1000,1000). The alignment was sorted, and a pileup file was generated for the alignment over all 4 PBP genes with SAMtools mpileup.

Custom code parses the pileup file to identify the variants that the HaplotypeCaller and Kestrel variant calls are compared against. For any loci with a depth other than 1, reliable variants cannot be extracted, and these are marked as no-call loci regardless of the HaplotypeCaller and Kestrel results.

4.4.3.2 Alignment

For HaplotypeCaller, reads are aligned to the TIGR4 reference with BWA mem. With Picard Tools, the alignment is sorted, duplicates are marked, and the alignment is sorted again (SortSam and MarkDuplicatesWithMateCigar). Indel realignment is then performed on the marked alignment with GATK (RealignerTargetCreator and IndelRealigner). Lastly, variants are called with GATK HaplotypeCaller over the PBP gene regions.

4.4.3.3 Kestrel

KAnalyze is first run on the sequence reads with a k-mer size of 31, a Phred scaled base quality filter of 30, and a minimum frequency of 5. An indexed k-mer count (IKC) file is generated using a minimizer size of 15. With the IKC file and the reference, Kestrel generates a set of sequence calls over the PBP genes. Kestrel can run in one step by reading the FASTQ files, but they were run as separate steps so that runtime performance could be measured for each one.

4.4.3.4 Comparing variant calls

To avoid incorrectly assigning false positive and false negative calls, the testing pipeline automatically accounts for differences in the way the same variant is annotated. HaplotypeCaller often represents dense SNPs as an insertion and a deletion, and so the number of expected true variants is lower for this approach because many SNPs can be represented in two indel events. Erroneous false positive and false negative calls would result if these indels are not reconciled with the SNPs identified by the assembly.

For each false positive variant call, the sequence of the variant with 5 bases flanking it on each end was determined. This was accomplished by taking the reference sequence and replaying the variant along with any neighboring variants that affect the 5 base flank. That sequence, including the flank, is then compared to the sequence of the affected gene as determined by the assembly. If the variant sequence fragment matched any part of the assembled gene sequence, then it was accepted as a true negative.

For each false negative call, a similar approach was taken, but the variants were replayed from the expected set of variants and compared to the sequence of the gene as determined by the approach being tested (alignment or Kestrel). If the sequence fragment from the assembly with the variant call matched the sequence as determined by the approach tested, then it was accepted as a true positive.

4.4.4 Variant call performance

For all regions where the alignment depth was 15 or greater, Kestrel identified all 5,290 true variants with 0 false positives. HaplotypeCaller identified all 5,273 true variants with 0 false positives. The number of expected variants differs as a side-effect of differences in variant annotation (Section 4.4.3.4).

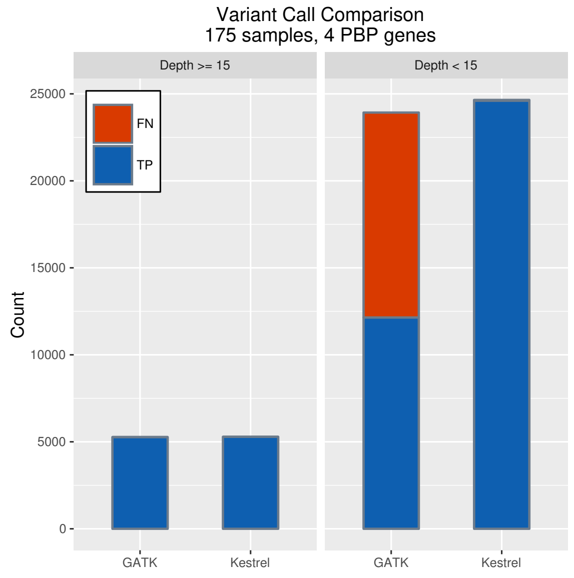

Figure 19: Variant calls from Kestrel and GATK are shown for regions with an alignment depth of at least 15 (left) and with less than 15 (right).

Where the alignment depth is less than 15x, Kestrel identified all 24,588 true variants with 2 false positives (Sensitivity = 1.000, False discovery rate (FDR) = 8.1334e-5). Because alignments are unreliable over these loci, HaplotypeCaller identified 12,078 of 23,855 true variants with 24 false positives (Sensitivity = 0.5063, FDR = 1.9831e-3) (Figure 19).

Figure 20 depicts the number of variant calls for each sample with the ANI phylogenetic tree for all samples. Some distantly related samples apparently contained mosaic regions characteristic of its lineage. Whether the mosaic region is part of the lineage or a *de novo* event, Kestrel is able to characterize it.

The HaplotypeCaller false positives were found where the alignment declined. Filtering by the lowest quality score of all of these calls, 60, would have changed 156 true positives to false negatives. Filtering by the lowest locus depth, 1, would have changed 5 true positives. Therefore, I did not attempt to apply quality or depth filters to improve the HaplotypeCaller results.

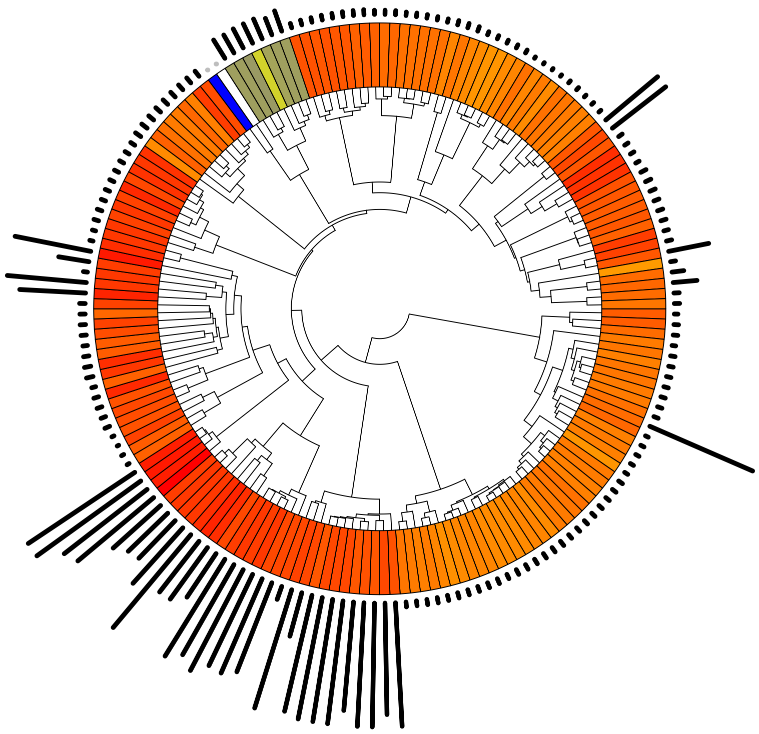

Figure 20: Whole genome phylogeny (ANI) (inner track), a heatmap showing the distance from the reference (white) where blue is closely related and red is distant (middle track), and a bar chart showing the number of variants per sample (outer track).

4.4.5 Runtime performance

All tests were run on a 12 core machine (2 x Intel Xeon E5-2620) with 32 GB of RAM (DDR3-1600), RAID-6 over SATA drives (3 GB/s, 72K RPM), and CentOS 6.7.

Kestrel required an average of 3.35 minutes per million 250 bp reads (Standard deviation (sd) = 0.74), the alignment approach required an average of 19.21 minutes per million 250 bp reads (sd = 3.72), and the assembly approach required 30.52 minutes per million 250 bp reads (sd = 12.53) (Figure 21(a)). Per sample, there is a linear relationship between the time Kestrel and the alignment pipeline required (R^2

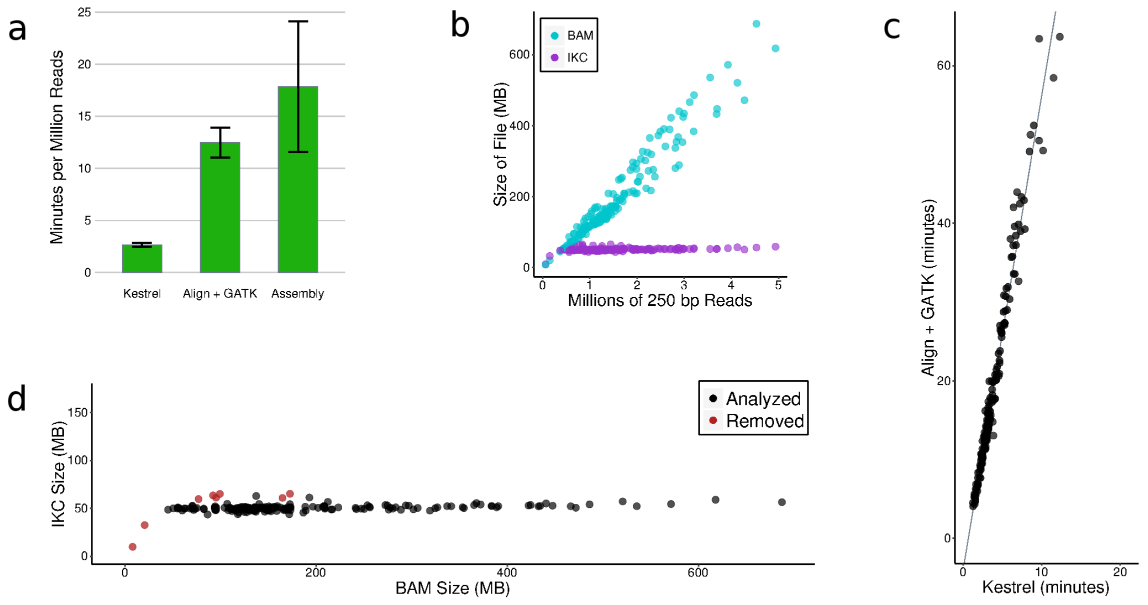

Figure 21: Kestrel performance by runtime and IKC file size compared with the alignment approach. (**a**) The mean time required by each approach per megabase (Mbase) of input. Error bars show a 95% confidence interval. (**b**) Comparison of BAM and IKC file file sizes as a function of the number of reads in a data set. (**c**) There is a linear relationship between the runtime for the alignment approach and the Kestrel in this data. (**d**) Size of BAM and IKC files per sample. Removed samples are shown in red.

= 0.80) with the alignment approach taking approximately 5.94x longer than Kestrel (Figure 21(c)). There is a noticeable skew along the Kestrel axis suggesting that some data sets may affect Kestrel's runtime disproportionately.

Unlike BAM files, IKC files do not grow with read depth because the number of unique k-mers is fixed in a sample (Figure 21(b)). When the IKC files contain more k-mers than expected, the number of unique k-mers, and therefore the file size, does increase (Figure 21(c)).

4.5 Discussion

The strength of Kestrel is its ability to identify variants where the sample differs greatly from the reference. The algorithm makes no *a priori* assumptions about the haplotypes over the active region other than that they should produce an acceptable alignment against the corresponding reference region given a set of scoring criteria.

In theory, Kestrel is capable of identifying arbitrarily large insertions. In practice, the alignment scores must be adjusted to a reasonable value, and this limits the size of insertions Kestrel will find. If the alignment could tolerate a 1 Kb insertion, then the algorithm will assemble haplotypes up to at least 1 Kb before giving up on an erroneous assembly. However, this approach may still be faster than performing a *de novo* assembly to identify large insertions.

One of the most significant limitations of Kestrel is its inability to properly handle regions that have significant homology with other regions. This limitation stems from the fact that the context of the whole read and of paired-end reads is lost when sequences are transformed to k-mer frequencies (Section 2.2.5). A sophisticated set of heuristics is implemented in Kestrel to address this problem, but it remains a difficult task without alignments or the paired assembly graphs implemented by SPAdes.

Despite its limitations, Kestrel can characterize large genomic events with minimal computing resources. This not only shortens the analysis time, but it also allows this approach to be scaled more efficiently. As high-volume sequencing technology becomes faster and cheaper, reducing the cost of data storage and analysis becomes critical [59, 107]. Not only does Kestrel require less memory and CPU time, storing IKC files requires less disk space than BAM files, and the size of IKC files does not increase with greater read depth as long as low frequency k-mers from sequencing errors are removed.

Very little research has been conducted where k-mers are used directly for variant calling. One notable exception, kSNP [39, 38], demonstrated that k-mers could call SNPs for phylogenetic analysis. A useful property of kSNP can compare samples with or without a reference sequence. This enables SNP-based phylogeny among samples where a reference is unavailable, but phylogeny can include a reference genome where it is available.

kSNP compares k-mers from samples and allows the central base to vary. If a k-mer from one sample matches a k-mer with a different central base in another sample, it is evidence that there is a SNP at that base. Given the k-mers of a reference sequence and the k-mers of a sample, it is trivial to see how this algorithm could function as a variant caller for SNPs. However, if another SNP occurs in close proximity, then the sequence flanking the central base is also altered, and both SNPs would be lost. Error correction in the form of a hamming distance could be employed, but this seems to quickly become an unwieldy and inefficient way to call SNPs when an alignment-based variant caller would work better. Furthermore, there is no clear way to identify even the smallest indel variants this way. kSNP is an excellent example of identifying variants using k-mers without de Bruijn graphs, but its approach would not scale well to identify a wider variety of events.

A subtle but important advancement of Kestrel is that it can utilize k-mers in a more flexible way. Recall from Section 2.3.1 that a significant advantage of k-mers is in their numeric representation. This view on k-mers is very rigid because two k-mers are either an exact match or no match at all regardless of how similar they may be. The alignment-guided reassembly process can incorporate k-mers into haplotypes that are arbitrarily distant from the reference sequence. The modified Smith-Waterman alignment algorithm is also unique in its ability to align one sequence from end to end and allow another sequence to extend arbitrarily over it. In my research, I have never encountered anything like it.

Kestrel is open source, and it designed with using best scientific computing practices [117, 112]. The software was built to be run by people and pipelines, and it was built as an application programming interface (API) so that it can be directly integrated into other projects. This implementation was made to be as flexible as possible so that the parts of Kestrel, such as the aligner, could be used independently by other types of applications.

CHAPTER V

APPLICATIONS OF KESTREL

5.1 Abstract

Throughout this dissertation on alternative approaches for sequence analysis, bacterial typing and drug resistance have been themes for the application of this technology. I have focused mainly on the algorithms themselves, but not their application. This chapter presents experiments done with Kestrel, and it discusses an application to human cancers.

The structure of this chapter varies from previous chapters. Instead of one story from introduction to discussion, three mini-chapters with those same sections are found here, and each tells one story about the application of Kestrel. The chapter will wrap up with a discussion on the future applications of Kestrel and k-mer approaches.

5.2 Applying Kestrel to human cancers

Kestrel has been shown to work on bacterial data. However, it is a haplotype caller, and it was built to handle human data. Some early development efforts on Kestrel were done with human sequences, and addressing the problems presented by those data made it a much stronger tool. Since I found a convincing story for this technology in *Streptococcus pneumoniae* (*S. pneumoniae*) PBP genes, that became the focus of the Kestrel publication ([**Submitted**]). The results of those early efforts are presented here.

5.2.1 Introduction

Mutations accumulate in tumors where some are biologically active and confer a growth advantage to the tumor [45, 97] or are responsible for chemotherapeutic resistance [42, 99]. BRAF$^{\mathrm{V600E}}$ appears in at least 40% of melanomas, and it drives cell proliferation by constitutively activating MEK/ERK pathway [3, 15].

Predicting drug sensitivity is a complicated task, and the best algorithms use multiple data sets from each tumor, such as gene expression profiles and variant calls in gene sequence data [23]. Here, I focus on extracting variant calls from sequence reads.

Kestrel was tested on sequence data from 526 cancer-associated exons published by Newman *et al.* [91]. These data, released in NCBI BioProject PRJNA241385, contain both a healthy control sample and a *non-small-cell lung carcinoma* (NSCLC) cell line sample.

The goal of that study was to develop a system for detecting circulating tumor DNA by sequencing samples with various dilutions of a tumor sample within a healthy control sample. I was originally interested to see if Kestrel could detect variants in the diluted tumor DNA. I found that it could call variants from a 1% dilution, but it would be too difficult to separate them from erroneous variants. For this discussion, I will use only the healthy control and the pure cancer sample for variant calling with Kestrel.

5.2.2 Methods and algorithms

The healthy control and the NSCLC samples from the Newman *et al.* [91] study were downloaded from NCBI BioProject PRJNA241385.

From their 526 exons, 14 were removed. One was a duplicate, and the remaining 13 shared significant homology with other regions of the genome. Kestrel can work around peaks that cover a few bases, but it cannot properly call variants where homology with other regions is too great (Section 4.5), and so they were removed. Several of these were olfactory receptor exons, which are known to appear as false positive associations in cancer informatic studies [66]. Others, such as PRSS1, have up to 18 paralogues according to Ensembl, and this is too much homology for the k-mer approach to handle confidently.

All variants wele called using a BWA [68, 69], Picard, and GATK [85] HaplotypeCaller pipeline. Variants were called with Kestrel using a k-mer size of 48. 31-mers were originally tested, however, this led to many false-positive results, and 48-mers proved to be adequate. The calls from both approaches were merged and analyzed. Discrepancies between the calls were analyzed. Some low quality calls were removed. A few exons had poor coverage, and therefore a low quality score, but the alignments were good for the reads that were present, and so the variants in these exons were not removed.

5.2.3 Results

Of the 70 high-quality variants in the healthy control, Kestrel correctly identified 68 (sensitivity = 0.97). Both of the missed variants were at the edge of an amplified exon where coverage was increasing dramatically, and the active region detector failed to find them. There were two false positives, but I could not find an explanation for them. The alignment pipeline completed in 360 minutes (178 to align, and 182 to call variants). Kestrel completed in 23 minutes (21 minutes to generate 48-mer counts, and 2 minutes to call variants). A speedup of 15.65 was achieved in this example.

Of the 96 high-quality variants in the NSCLC sample, Kestrel identified 92 (sensitivity = 0.96), and there were no false positives. 3 of the 4 missed variants were in a region that was not amplified well and had extremely low coverage. GATK confidently called 2 false positives. These false positives were in regions that were deeply sequenced, and I could not find an explanation for why GATK would call them. The alignment pipeline completed in 95 minutes (87 to align, and 8 to call variants). Kestrel completed in 16 minutes (15 minutes to generate 48-mer counts, and 1 minute to call variants). A speedup of 5.94 was achieved in this example.

5.2.4 Discussion

This is a limited example of how Kestrel can work on human amplicon data. I found that a k-mer size of 48 worked well, but it could not eliminate all homology from paralogues in k-mer space, which would have required a k-mer size larger than the read size. This recapitulates one advantage of alignments over k-mers: mapping paired-end reads can resolve much of this homology, but paired-end context is lost when reads are converted to k-mer frequencies.

Despite the limitations, Kestrel agreed with GATK for most variants, however, it produced an answer many times faster. When applied correctly and with care, Kestrel could be used on human data, but it is not well suited as a general purpose variant caller in this context.

5.3 TB drug resistance

Section 4.4 showed that Kestrel could call variants within gene regions, but it is also capable of processing whole genomes. The experiment described here performs variant calling and the whole *Mycobacterium tuberculosis* (*M. tuberculosis*) 4.4 Mb reference and reveals known patterns of drug resistance.

5.3.1 Introduction

Unlike *S. pneumoniae*, *M. tuberculosis* lacks natural competence, and therefore the ability to take up and incorporate foreign DNA into its genome. Some large variants do exist in this species, but they are not as prolific. Variant calling does not require an advanced tool like Kestrel, but it may still benefit from an increased speed of analysis.

Two *Mycobacterium tuberculosis* (*M. tuberculosis*) samples were obtained from Török *et al.* [31] with known variants in drug resistance genes, and 90 samples from Farhat *et al.* [33]. The Török study used a corrected *M. tuberculosis* H37Rv reference [16], and we used the same reference to ensure that our results were comparable with the known variants.

5.3.2 Methods and algorithms

All variants were called against the corrected *M. tuberculosis* H37Rv reference [16], and variants were filtered for SNPs. Variants in all members of the repetitive PE/PPE class of genes were filtered out.

Sequences were processed by KAnalyze into a 31-mer IKC file with a minimum frequency of 5, and a minimum allele depth of 0.50. Custom Python code reads the reference and translates variants to their amino-acid changes.

Many rpoB amino-acid changes are published with coordinates relative to the *Escherichia coli* (*E. coli*) rpoB gene. Using a global alignment of the rpoB protein sequence from the H37Rv reference and an *E. coli* k12 reference, custom code was created to translate coordinates between the two references. With this translation, I was able to correlate variants from the H37Rv reference with published data and determine which were responsible for rifampicin resistance.

5.3.3 Results

112,463 occurrences of 10,076 unique SNPs were identified in all 92 samples. As expected, all 13 SNPs in published in the Török study were found in those two samples.

katG and rpoB were further analyzed for patterns of resistance to isoniazid and rifamicin, respectively. Within these two genes, 119 occurrences of 21 unique SNPs was found in all 92 samples. 11 of these SNPs were non-synonymous, and 8 of those likely confer drug resistance. Four unique SNPs were found in the *rifampicin* *resistance-determining region* (RRDR) of rpoB. Table 1 shows the variants in these genes. Figure 22 shows the locations of the katG and rpoB variants.

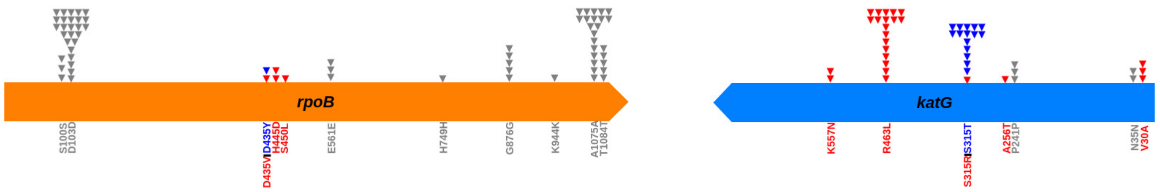

Figure 22: Location of variants in the *M. tuberculosis* rpoB and katG genes. Non-silent mutations are in red and blue, and silent mutations are gray.

5.3.4 Discussion

This project illustrates Kestrel's ability to identify variants with known associations. Because of the speed of variant calling, as demonstrated by previous projects, Kestrel could be easily integrated into surveillance pipelines for diseases like Tuberculosis.

Converting the sequence reads to k-mers often takes more time than running the analysis from the k-mer frequencies, as illustrated by previous examples (Section 5.2.3). What is interesting is that different types of analysis could be done on this data. Kestrel shows that variants can be called from it, but it was also shown that informatic spoligotyping can be done as well (CHAPTER III). Kestrel could ostensibly build the entire CRISPR locus with a larger k-mer size. For example, a 48-mer might be large enough to span the 36 bp repeats. From 48-mer frequencies, typing might be done by examining the CRISPR spacers or by SNP typing for higher resolution. Other actionable variants, such as those associated with drug resistance, can also be identified. Since all this can be done faster than it could with alignments, a pathogen surveillance platform may output more information with less time and lower cost.

Table 1: SNPs found in katG (Rv1908c) and rpoB (Rv0667) from all 92 samples. Variants are relative to genes annotated in the corrected H37Rv reference, and rpoB variants in parenthesis are relative to the *E. coli* K12 rpoB reference. Synonymous SNPs are colored gray. SNPs in the RRDR of rpoB are marked with a dagger (†).

Location	Ref	Alt	Gene	Variant	Count	Resistance
2156031	A	G	katG	V30A	4	
2156015	G	A	katG	N35N	2	
2155397	G	C	katG	P241P	4	
2155354	C	T	katG	A256T	1	Likely [35, 113]
2155175	G	T	katG	S315R	1	Likely [90]
2155176	C	G	katG	S315T	16	Likely [90, 104, 88]
2154732	C	A	katG	R463L	18	Likely [90, 104, 84]
2154449	C	G	katG	K557N	2	
760111	G	A	rpoB	S100S (S69S)	4	
760120	C	T	rpoB	D103D (D72D)	28	
761115	A	T	rpoB	D435V (D516V)†	1	Yes [106, 29]
761114	G	T	rpoB	D435Y (D516Y)†	2	Yes [106]
761144	C	G	rpoB	H445D (H526D)†	2	Yes [106]
761160	C	T	rpoB	S450L (S531L)†	1	Yes [106, 29]
763063	C	T	rpoB	I491M (I572M)	1	Likely [109]
761494	G	A	rpoB	E561E (E641E)	3	
762058	T	C	rpoB	H749H (H836H)	2	
762439	T	G	rpoB	G876G (G1063G)	5	
762643	G	A	rpoB	K944K (K1163K)	1	
763036	T	C	rpoB	A1075A (A1283A)	18	
763063	C	T	rpoB	T1084T (T1292T)	3	

5.4 MLST typing

Section 3 discusses typing and presents a method for informatic spoligotyping. However, other methods of typing can be performed with these approaches. This section discusses a project where *multilocus sequence typing* (MLST) was performed with

Kestrel.

5.4.1 Introduction

Multilocus sequence typing (MLST) [78] has the power to resolve typing with great detail. This scheme was originally developed on stable house-keeping genes, but for greater resolution, it can be applied to genes that exhibit more diversity, and therefore, a greater seletion of alleles among individuals [79].

The current method for performing MLST on WGS data is to execute a *de novo* assembly on the sequence data and to use BLAST [2] comparing known alleles with the assemblies [65, 54]. Because it requires both assemblies an alignments, this process can take a long time to run on a sample.

I was interested to see if MLST could be done without an assembly or BLAST using Kestrel. With my guidance, Shashidhar Ravishankar, a fellow Ph.D. student, designed and executed the experiment. The following results are from his work, and he is currently improving this method so that it scales over more genes and more alleles.

5.4.2 Methods and algorithms

7 *Neisseria meningitidis* (*N. meningitidis*) samples were obtained from from ENA study PRJEB3353 (ERR193671-ERR193677) [100] and allele sequences for 7 house-keeping genes (adk, aroE, abcZ, fumC, gdh, pgm, and pdhC) from pubMLST [54]. These data are WGS 150 bp paired-end Illumina reads.

For each sample, the KAnalyze [4] API transforms sequence data to k-mer frequencies using 31-mers and writes an IKC file with a 15 bp minimizer.

For each gene in the MLST scheme, the Kestrel algorithm is applied using each allele as a reference and requiring a minimum allele depth of 0.50. The best allele has the fewest variants and a minimum k-mer frequency greater than 0. Prototype Python scripts read the output from this step, merge results, and output final MLST calls. Lastly, the best allele matches were used to identify the sequence type by comparing them against types from pubMLST.

The current standard method was executed to veryify the MLST calls. Sequence reads were assembled with SPAdes [5] using default options, and scaffolds were used to construct a BLAST database. Each allele downloaded from pubMLST for *N. meningitidis* was used as a BLAST query, and the best allele for each gene was chosen. The best allele produced the lowest E value when aligned with the scaffolds by BLAST. With the allele calls, the sequence type (ST) was identified by comparing against sequence type profiles also obtained from pubMLST. This set of alleles and the sequence type represents the expected results based on the current methods.

5.4.3 Results

There was 100% concordance between the two methods. Table 2 shows allele and sequence types for each sample.

All samples were called with a minimum k-mer frequency of 5 except ERR193672, which was reduced to 2 to call gene pgm. The assembly method called this allele as confidently as the other alleles with higher coverage.

5.4.4 Discussion

While this led to a far more efficient approach without *de novo* assembly, there are improvements currently in development. Instead of applying Kestrel to the individual allele sequences, it can be applied to a consensus sequence. Then, each allele can be compared more efficiently in $\mathcal{O}(\log n)$ time. These improvements will allow MLST typing scale to many genes and thousands of alleles while making the most of computing resources. Shashidhar Ravishankar is actively developing this technique.

5.5 Future applications

5.5.1 Whole genome SNP typing

Many of the techniques discussed, such as MLST and spoligotyping, sample a small subset of the genome. These methods have proven useful for grouping clinically relevant individuals, which is why they are still in use today. For finer resolution of samples, using a much larger subset of the genome captures more variation.

An interesting example comes from a study of methicillin-resistant *Staphylococcus aureus* (MRSA) [47]. 20 of the samples were closely related because they were isolated from the same hospital over a 7 month period. 5 of these 20 showed strong evidence for nosocomial transmission within the hospital itself. This is clearly a clinically relevant finding, and it could not have been identified with lower-resolution methods.

Because Kestrel can rapidly call SNPs on a whole genome, it is possible that it can replace alignment-based variant callers for phylogenetic analysis, and this can be particularly useful for organisms that exhibit significant levels of horizontal gene transfer (HGT), such as *S. pneumoniae*.

When working with organisms that are capable of HGT, it is important to remember that one recombination event can be seen as a region of dense variation. In a SNP phylogeny study of *S. pneumoniae* [24], these events were collapsed. This corrected the length of the branches in the phylogenetic tree and reduced homoplastic sites to loci mostly occurring in genes associated with drug resistance. The MRSA study [47] was able to use the core genome to avoid this bias. This kind of post-SNP analysis would need to be done regardless of the SNP calling method.

5.5.2 K-mer phylogeny

Another whole genome approach is to attempt to compare the genomic content of each sample more directly. While SNPs are generally called from a reference for each sample, this approach compares each sample to each other sample for a total of $\frac{n \cdot (n-1)}{2}$

comparisons. These methods do not typically require a reference sequence, which may be useful for studies on organisms where the genome has not been resolved.

Average nucleotide identity (ANI) [44] is one method for doing this. It requires a *de novo* assembly for each sample, and it uses BLAST or another alignment method to compare the nucleotides in assemblies for each pair of samples. Like many other methods using *de novo* assemblies or alignments, this too can be done faster without alignments or assemblies. Instead, the k-mers of each sample can be compared for differences. These methods would not account for HGT events, and so the results may be biased for some organisms.

Table 2: Allele calls for all 7 genes in all 7 samples as called by the *de novo* approach and the Kestrel approach. The depth column shows the minimum depth used to identify the allele. The bolded rows show the sequence type (ST) for the sample, and its depth is the least of all allele depths.

ERR193671			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	11	11	5
adk	5	5	5
aroE	18	18	5
fumC	8	8	5
gdh	11	11	5
pdhC	4	4	5
pgm	21	21	5
ST	**184**	**184**	**5**

ERR193675			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	3	3	5
adk	6	6	5
aroE	9	9	5
fumC	5	5	5
gdh	9	9	5
pdhC	6	6	5
pgm	9	9	5
ST	**41**	**41**	**5**

ERR193672			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	10	10	5
adk	5	5	5
aroE	18	18	5
fumC	9	9	5
gdh	11	11	5
pdhC	9	9	5
pgm	12	12	2
ST	**4183**	**4183**	**2**

ERR193676			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	4	4	5
adk	4	4	5
aroE	2	2	5
fumC	5	5	5
gdh	38	38	5
pdhC	11	11	5
pgm	16	16	5
ST	**NA**	**NA**	**5**

ERR193673			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	420	420	5
adk	5	5	5
aroE	173	173	5
fumC	90	90	5
gdh	11	11	5
pdhC	24	24	5
pgm	21	21	5
ST	**11160**	**11160**	**5**

ERR193677			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	3	3	5
adk	6	6	5
aroE	9	9	5
fumC	5	5	5
gdh	8	8	5
pdhC	6	6	5
pgm	9	9	5
ST	**485**	**485**	**5**

ERR193673			
Gene	**De novo**	**Kestrel**	**Depth**
abcZ	9	9	5
adk	6	6	5
aroE	2	2	5
fumC	9	9	5
gdh	9	9	5
pdhC	6	6	5
pgm	9	9	5
ST	**2487**	**2487**	**5**

CHAPTER VI

CONCLUDING REMARKS

In CHAPTER II, I outlined the algorithms and data structures that are critical for analyzing sequence reads without sequence read alignments, *de novo* assembly, or de Bruijn graphs. KAnalyze, which implements these algorithms and structures, was discussed in detail. The advancements discussed in subsequent chapters are made possible by KAnalyze.

In CHAPTER III, I discuss an alignment-free approach for identifying the CRISPR spacers in *Mycobacterium tuberculosis* (*M. tuberculosis*) sequence data that resulted in a 22x reduction in runtime over existing methods. The error correcting algorithm described in this chapter was inefficient, and efforts to improve it led to the novel methods described in In CHAPTER IV.

In CHAPTER IV, I describe a novel variant calling software, Kestrel, that uses k-mer frequencies to identify variant regions and find variants within those regions using a haplotype approach. In *Streptococcus pneumoniae* (*S. pneumoniae*), Kestrel can reduce analysis time from 19.21 to 3.35 minutes per million reads while using less than 2GB of RAM. In the time it takes for the alignment approach to complete one sample, Kestrel can process more than 5.

In CHAPTER V, I discuss other projects using Kestrel and KAnalyze. These projects illustrate new potential uses of the technology and future applications.

Algorithms that do not rely on sequence read alignments, *de novo* assemblies, or de Brujin graphs can analyze genomic data faster than the current state of the art.

REFERENCES

[1] "Global Tuberculosis Report," tech. rep., World Health Organization, 2015.

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J., "Basic Local Alignment Search Tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, oct 1990.

[3] Ascierto, P. a., Kirkwood, J. M., Grob, J.-J., Simeone, E., Grimaldi, A. M., Maio, M., Palmieri, G., Testori, A., Marincola, F. M., and Mozzillo, N., "The role of BRAF V600 mutation in melanoma," *Journal of Translational Medicine*, vol. 10, no. 1, p. 85, 2012.

[4] Audano, P. and Vannberg, F., "KAnalyze: a fast versatile pipelined K-mer toolkit," *Bioinformatics*, vol. 30, pp. 2070–2, July 2014.

[5] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., and Pevzner, P. A., "SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing," *Journal of Computational Biology*, vol. 19, no. 5, pp. 455–477, 2012.

[6] Barrangou, R., Fremaux, C., Deveau, H., Richards, M., Boyaval, P., Moineau, S., Romero, D. A., and Horvath, P., "CRISPR Provides Acquired Resistance Against Viruses in Prokaryotes," *Science*, vol. 315, pp. 1709–1712, mar 2007.

[7] Bentley, D., Balasubramanian, S., Swerdlow, H., Smith, G., Milton, J., Brown, C., Hall, K., Evers, D., Barnes, C., Bignell, H., Boutell, J., Bryant, J., Carter, R., Keira Cheetham, R., Cox, A., Ellis, D., Flatbush, M., Gormley, N., Humphray, S., Irving, L., Karbelashvili, M., Kirk, S., Li, H., Liu, X., Maisinger, K., Murray, L., Obradovic, B., Ost, T., Parkinson, M., Pratt, M., Rasolonjatovo, I. M., Reed, M., Rigatti, R., Rodighiero, C., Ross, M., Sabot, A., Sankar, S., Scally, A., Schroth, G., Smith, M., Smith, V., Spiridou, A., Torrance, P., Tzonev, S., Vermaas, E., Walter, K., Wu, X., Zhang, L., Alam, M., Anastasi, C., Aniebo, I., Bailey, D. M., Bancarz, I., Banerjee, S., Barbour, S., Baybayan, P., Benoit, V., Benson, K., Bevis, C., Black, P., Boodhun, A., Brennan, J., Bridgham, J., Brown, R., Brown, A., Buermann, D., Bundu, A., Burrows, J., Carter, N. P., Castillo, N., Chiara E Catenazzi, M., Chang, S., Neil Cooley, R., Crake, N., Dada, O., Diakoumakos,

K., Dominguez-Fernandez, B., Earnshaw, D., Egbujor, U., Elmore, D., Etchin, S., Ewan, M., Fedurco, M., Fraser, L., Fuentes Fajardo, K., Scott Furey, W., George, D., Gietzen, K., Goddard, C., Golda, G., Granieri, P., Green, D., Gustafson, D., Hansen, N., Harnish, K., Haudenschild, C., Heyer, N., Hims, M., Ho, J., Horgan, A., Hoschler, K., Hurwitz, S., Ivanov, D., Johnson, M., James, T., Huw Jones, T. A., Kang, G., Kerelska, T., Kersey, A., Khrebtukova, I., Kindwall, A., Kingsbury, Z., Kokko-Gonzales, P., Kumar, A., Laurent, M., Lawley, C., Lee, S., Lee, X., Liao, A., Loch, J., Lok, M., Luo, S., Mammen, R., Martin, J., Mccauley, P., Mcnitt, P., Mehta, P., Moon, K., Mullens, J., Newington, T., Ning, Z., Ling Ng, B., Novo, S., O'Neill, M. J., Osborne, M., Osnowski, A., Ostadan, O., Paraschos, L., Pickering, L., Pike, A., Chris Pinkard, D., Pliskin, D., Podhasky, J., Quijano, V., Raczy, C., Rae, V., Rawlings, S., Chiva Rodriguez, A., Roe, P., Rogers, J., Rogert Bacigalupo, M., Romanov, N., Romieu, A., Roth, R., Rourke, N., Ruediger, S., Rusman, E., Sanches-Kuiper, R., Schenker, M., Seoane, J., Shaw, R., Shiver, M., Short, S., Sizto, N., Sluis, J., Ernest Sohna Sohna, J., Spence, E., Stevens, K., Sutton, N., Szajkowski, L., Tregidgo, C., Turcatti, G., Vandevondele, S., Verhovsky, Y., Virk, S., Wakelin, S., Walcott, G., Wang, J., Worsley, G., Yan, J., Yau, L., Zuerlein, M., Mullikin, J., Hurles, M. E., Mccooke, N., West, J., Oaks, F., Lundberg, P., Klenerman, D., Durbin, R., and Smith, A., "Accurate whole human genome sequencing using reversible terminator chemistry," *Nature*, vol. 456, no. 7218, pp. 53–59, 2008.

[8] Bertelli, C. and Greub, G., "Rapid bacterial genome sequencing: methods and applications in clinical microbiology," *Clinical Microbiology and Infection*, vol. 19, pp. 803–813, sep 2013.

[9] Blair, J. E. and Carr, M., "The bacteriophage typing of staphylococci," *Journal of Infectious Diseases*, vol. 93, no. 1, pp. 1–13, 1953.

[10] Bloom, B., "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *ACM Communications*, vol. 13, no. 7, 1970.

[11] Borrell, S. and Gagneux, S., "Infectiousness, reproductive fitness and evolution of drug-resistant Mycobacterium tuberculosis," *International Journal of Tuberculosis and Lung Disease*, vol. 13, no. 12, pp. 1456–1466, 2009.

[12] Bray, N. L., Pimentel, H., Melsted, P., and Pachter, L., "Near-optimal probabilistic RNA-seq quantification," *Nature Biotechnology*, vol. 34, no. 5, pp. 525–527, 2016.

[13] Burrows, M. and Wheeler, D., "A block-sorting lossless data compression algorithm," Tech. Rep. 124, Digital Equipment Corporation. CA, 1994.

[14] Callow, B. R., "Bacteriophage Phenomena with Staphylococcus Aureus," *Journal of Infectious Diseases*, vol. 30, pp. 643–650, 1922.

[15] Cantwell-Dorris, E. R., O'Leary, J. J., and Sheils, O. M., "BRAFV600E: implications for carcinogenesis and molecular therapy," *Mol Cancer Ther*, vol. 10, no. 3, pp. 385–394, 2011.

[16] Casali, N., Nikolayevskyy, V., Balabanova, Y., Ignatyeva, O., Kontsevaya, I., Harris, S. R., Bentley, S. D., Parkhill, J., Nejentsev, S., Hoffner, S. E., Horstmann, R. D., Brown, T., and Drobniewski, F., "Microevolution of extensively drug-resistant tuberculosis in Russia," *Genome Research*, vol. 22, pp. 735–745, apr 2012.

[17] Chee, C. B., Gan, S.-H., Ong, R. T., Sng, L.-H., Wong, C. W., Cutter, J., Gong, M., Seah, H.-M., Hsu, L. Y., Solhan, S., Ooi, P.-L., Xia, E., Lim, J. T., Koh, C.-K., Lim, S.-K., Lim, H.-K., and Wang, Y.-T., "Multidrug-Resistant Tuberculosis Outbreak in Gaming Centers, Singapore, 2012," *Emerging Infectious Diseases*, vol. 21, no. 1, pp. 179–180, 2015.

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

[19] Chor, B., Horn, D., Goldman, N., Levy, Y., and Massingham, T., "Genomic DNA k-mer spectra: models and modalities.," *Genome biology*, vol. 10, no. 10, p. R108, 2009.

[20] Coll, F., Mallard, K., Preston, M. D., Bentley, S., Parkhill, J., McNerney, R., Martin, N., and Clark, T. G., "SpolPred: rapid and accurate prediction of Mycobacterium tuberculosis spoligotypes from short genomic sequences," *Bioinformatics*, vol. 28, no. 22, pp. 2991–2993, 2012.

[21] Consortium, T. . G. P., "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, pp. 68–74, 2015.

[22] Consortium, T. I. H., "The International HapMap Project," *Nature*, vol. 426, no. 6968, pp. 789–796, 2003.

[23] Costello, J. C., Heiser, L. M., Georgii, E., Gönen, M., Menden, M. P., Wang, N. J., Bansal, M., Ammad-Ud-Din, M., Hintsanen, P., Khan, S. A., Mpindi, J.-P., Kallioniemi, O., Honkela, A., Aittokallio, T., Wennerberg, K., Collins, J. J., Gallahan, D., Singer, D., Saez-Rodriguez, J., Kaski, S., Gray, J. W., and Stolovitzky, G., "A community effort to assess and improve drug sensitivity prediction algorithms," *Nature Biotechnology*, vol. 32, no. 12, pp. 1202–1214, 2014.

[24] Croucher, N. J., Harris, S. R., Fraser, C., Quail, M. A., Burton, J., van der Linden, M., McGee, L., von Gottberg, A., Song, J. H., Ko, K. S., Pichon, B., Baker, S., Parry, C. M., Lambertsen, L. M.,

Shahinas, D., Pillai, D. R., Mitchell, T. J., Dougan, G., Tomasz, A., Klugman, K. P., Parkhill, J., Hanage, W. P., and Bentley, S. D., "Rapid Pneumococcal Evolution in Response to Clinical Interventions," *Science*, vol. 331, pp. 430–434, jan 2011.

[25] De Beenhouwer, H., Lhiang, Z., Jannes, G., Mijs, W., Machtelinckx, L., Rossau, R., Traore, H., and Portaels, F., "Rapid detection of rifampicin resistance in sputum and biopsy specimens from tuberculosis patients by PCR and line probe assay," *Tubercle and Lung Disease*, vol. 76, no. 5, pp. 425–430, 1995.

[26] Doughty, E. L., Sergeant, M. J., Adetifa, I., Antonio, M., and Pallen, M. J., "Culture-independent detection and characterisation of Mycobacterium tuberculosis and M. africanum in sputum samples using shotgun metagenomics on a benchtop sequencer," *PeerJ*, vol. 2, sep 2014.

[27] Eddy, S. R., "What is dynamic programming?," *Nature Biotechnology*, vol. 22, no. 7, pp. 909–910, 2004.

[28] Editor, "Gathering clouds and a sequencing storm," *Nature Biotechnology*, vol. 28, no. 1, p. 1, 2010.

[29] Eldholm, V., Monteserin, J., Rieux, A., Lopez, B., Sobkowiak, B., Ritacco, V., and Balloux, F., "Four decades of transmission of a multidrug-resistant Mycobacterium tuberculosis outbreak strain," *Nature Communications*, vol. 6, no. May, p. 7119, 2015.

[30] Embden, J. D. a. V., Gorkom, T. V., Kremer, K., Jansen, R., Zeijst, B. a. M. V. D., and Schouls, L. M., "Genetic Variation and Evolutionary Origin of the Direct Repeat Locus of Mycobacterium tuberculosis Complex Bacteria," *Journal of Bacteriology*, vol. 182, no. 9, pp. 2393–2401, 2000.

[31] Estée Török, M., Reuter, S., Bryant, J., Köser, C. U., Stinchcombe, S. V., Nazareth, B., Ellington, M. J., Bentley, S. D., Smith, G. P., Parkhill, J., and Peacock, S. J., "Rapid Whole-Genome Sequencing for Investigation of a Suspected Tuberculosis Outbreak," *Journal of Clinical Microbiology*, vol. 51, no. 2, pp. 611–614, 2013.

[32] European Concerted Action on New Generation Genetic Markers and Techniques for the Epidemiology and Control of Tuberculosis, "Beijing/W Genotype Mycobacterium tuberculosis and Drug Resistance," *Emerging Infectious Diseases*, vol. 12, no. 5, pp. 736–743, 2006.

[33] Farhat, M. R., Shapiro, B. J., Kieser, K. J., Sultana, R., Jacobson, K. R., Victor, T. C., Warren, R. M., Streicher, E. M., Calver, A., Sloutsky, A., Kaur, D., Posey, J. E., Plikaytis, B., Oggioni, M. R., Gardy, J. L., Johnston, J. C., Rodrigues, M., Tang, P. K. C., Kato-Maeda, M., Borowsky, M. L., Muddukrishna, B., Kreiswirth, B. N.,

Kurepina, N., Galagan, J., Gagneux, S., Birren, B., Rubin, E. J., Lander, E. S., Sabeti, P. C., and Murray, M., "Genomic analysis identifies targets of convergent positive selection in drug-resistant Mycobacterium tuberculosis," *Nature Genetics*, vol. 45, no. 10, pp. 1183–9, 2013.

[34] Feikin, D. R., Feldman, C., Schuchat, A., and Janoff, E. N., "Global strategies to prevent bacterial pneumonia in adults with HIV disease.," *The Lancet. Infectious diseases*, vol. 4, no. 7, pp. 445–55, 2004.

[35] Fenner, L., Egger, M., Bodmer, T., Altpeter, E., Zwahlen, M., Jaton, K., Pfyffer, G. E., Borrell, S., Dubuis, O., Bruderer, T., Siegrist, H. H., Furrer, H., Calmy, A., Fehr, J., Stalder, J. M., Ninet, B., Böttger, E. C., and Gagneux, S., "Effect of Mutation and Genetic Background on Drug Resistance in Mycobacterium tuberculosis," *Antimicrobial Agents and Chemotherapy*, vol. 56, no. 6, pp. 3047–3053, 2012.

[36] Ferragina, P. and Manzini, G., "Opportunistic data structures with applications," *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 390–398, 2000.

[37] Ford, C. B., Shah, R. R., Maeda, M. K., Gagneux, S., Murray, M. B., Cohen, T., Johnston, J. C., Gardy, J., Lipsitch, M., and Fortune, S. M., "Mycobacterium tuberculosis mutation rate estimates from different lineages predict substantial differences in the emergence of drug-resistant tuberculosis," *Nature genetics*, vol. 45, no. 7, pp. 784–790, 2013.

[38] Gardner, S. N. and Hall, B. G., "When Whole-Genome Alignments Just Won't Work: kSNP v2 Software for Alignment-Free SNP Discovery and Phylogenetics of Hundreds of Microbial Genomes," *PloS One*, vol. 8, p. e81760, jan 2013.

[39] Gardner, S. N. and Slezak, T., "Scalable SNP Analyses of 100+ Bacterial or Viral Genomes," *Journal of Forensic Research*, vol. 1, no. 2, p. 107, 2010.

[40] Garrison, E. and Marth, G., "Haplotype-based variant detection from short-read sequencing," *arXiv*, p. 9, 2012.

[41] GenomeWeb, "CDC Earmarks $2.3M for NGS, Bioinformatic Approaches to Combat Infectious Disease," 2015. Accessed May 31, 2016.

[42] Ghavami, S., Hashemi, M., Ande, S. R., Yeganeh, B., Xiao, W., Eshraghi, M., Bus, C. J., Kadkhoda, K., Wiechec, E., Halayko, a. J., and Los, M., "Apoptosis and cancer: mutations within caspase genes.," *Journal of medical genetics*, vol. 46, no. 8, pp. 497–510, 2009.

[43] Goldblatt, D., Rorman, E., Chemtob, D., Freidlin, P. J., Cedar, N., Kaidar-Shwartz, H., Dveyrin, Z., and Mor, Z., "Molecular epidemiology and mapping of tuberculosis in Israel: do migrants transmit the disease to

locals?," *The International Journal of Tuberculosis and Lung Disease*, vol. 18, pp. 1085–1091, sep 2014.

[44] Goris, J., Konstantinidis, K. T., Klappenbach, J. A., Coenye, T., Vandamme, P., and Tiedje, J. M., "DNA-DNA hybridization values and their relationship to whole-genome sequence similarities," *International Journal of Systematic and Evolutionary Microbiology*, vol. 57, no. 1, pp. 81–91, 2007.

[45] Greenman, C., Stephens, P. R., Bignell, G., Birney, E., Stratton, M. R., Smith, R. M., Dalgliesh, G., Hunter, C., Davies, H., Teague, J., Butler, A., Stevens, C., Edkins, S., O'Meara, S., Vastrik, I., Schmidt, E., Avis, T., Barthorpe, S., Bhamra, G., Buck, G., Choudhury, B., Clements, J., Cole, J., Dicks, E., Forbes, S., Gray, K., Halliday, K., Harrison, R., Hills, K., Hinton, J., Jenkinson, A., Jones, D., Menzies, A., Mironenko, T., Perry, J., Raine, K., Richardson, D., Shepherd, R., Small, A., Tofts, C., Varian, J., Webb, T., West, S., Widaa, S., Yates, A., Cahill, D., Louis, D., Goldstraw, P., Nicholson, A., Brasseur, F., Looijenga, L., Weber, B., Chiew, Y., DeFazio, A., Greaves, M., Green, A., Campbell, P., Easton, D., Chenevix-Trench, G., Tan, M., Khoo, S., Teh, B., Yuen, S., Leung, S., Wooster, R., and Futreal, P., "Patterns of somatic mutation in human cancer genomes," *Nature*, vol. 446, no. 7132, pp. 153–158, 2007.

[46] Hakenbeck, R., Kaminski, K., König, A., Van der linden, M., Paik, J., Reichmann, P., and Zähner, D., "Penicillin-Binding Proteins in β-LactamResistant Streptococcus pneumoniae," *Microbial Drug Resistance*, vol. 5, pp. 91–99, jan 1999.

[47] Harris, S. R., Feil, E. J., Holden, M. T. G., Quail, M. A., Nickerson, E. K., Chantratita, N., Gardete, S., Tavares, A., Day, N., Lindsay, J. A., Edgeworth, J. D., de Lencastre, H., Parkhill, J., Peacock, S. J., and Bentley, S. D., "Evolution of MRSA During Hospital Transmission and Intercontinental Spread," *Science*, vol. 327, pp. 469–474, jan 2010.

[48] Hasman, H., Saputra, D., Sicheritz-Ponten, T., Lund, O., Svendsen, C. A., Frimodt-Møller, N., and Aarestrup, F. M., "Rapid Whole-Genome Sequencing for Detection and Characterization of Microorganisms Directly from Clinical Samples," *Journal of Clinical Microbiology*, vol. 52, no. 1, pp. 139–146, 2014.

[49] Huse, S. M., Huber, J. A., Morrison, H. G., Sogin, M. L., and Welch, D. M., "Accuracy and quality of massively parallel DNA pyrosequencing," *Genome Biology*, vol. 8, no. 7, p. R143, 2007.

[50] Illumina, "A comparison of illumina sequencing platforms.," 2016. http://www.illumina.com/systems/sequencing.html [Accessed March 17, 2016].

[51] Institute, B., "Hc overview: How the haplotypecaller works," 2014. Accessed May 24, 2016.

[52] Institute, B., "Picard tools," 2016. Accessed April 22, 2016.

[53] Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., and McVean, G., "De novo assembly and genotyping of variants using colored de Bruijn graphs," *Nature genetics*, vol. 44, pp. 226–32, feb 2012.

[54] Jolley, K. A. and Maiden, M. C. J., "BIGSdb: Scalable analysis of bacterial genome variation at the population level," *BMC Bioinformatics*, vol. 11, no. 1, p. 595, 2010.

[55] Kamerbeek, J., Schouls, L., Kolk, A., van Agterveld, M., Kuijper, S., Bunschoten, A., Molhuizen, H., Shaw, R., Goyal, M., and van Embden, J., "Simultaneous detection and strain differentiation of Mycobacterium tuberculosis for diagnosis and epidemiology," *Journal of Clinical Microbiology*, vol. 35, no. 4, pp. 907–914, 1997.

[56] Knuth, D., *The Art of Computer Programming*, vol. 3. Addison-Wesley, 2nd ed., 1998. pp. 248-379.

[57] Kohl, T. a., Diel, R., Harmsen, D., Rothgänger, J., Meywald Walter, K., Merker, M., Weniger, T., and Niemann, S., "Whole-Genome-Based Mycobacterium tuberculosis Surveillance: A Standardized, Portable, and Expandable Approach," *Journal of Clinical Microbiology*, vol. 52, no. 7, pp. 2479–2486, 2014.

[58] Köser, C. U., Bryant, J. M., and Becq, J., "Whole-Genome Sequencing for Rapid Susceptibility Testing of M. tuberculosis," *New England Journal of Medicine*, vol. 369, pp. 290–292, jul 2013.

[59] Köser, C. U., Ellington, M. J., Cartwright, E. J. P., Gillespie, S. H., Brown, N. M., Farrington, M., Holden, M. T. G., Dougan, G., Bentley, S. D., Parkhill, J., and Peacock, S. J., "Routine Use of Microbial Whole Genome Sequencing in Diagnostic and Public Health Microbiology," *PLoS Pathogens*, vol. 8, no. 8, 2012.

[60] Kundeti, V. K., Rajasekaran, S., Dinh, H., Vaughn, M., and Thapar, V., "Efficient parallel and out of core algorithms for constructing large bi-directed de Bruijn graphs.," *BMC bioinformatics*, vol. 11, no. 1, p. 560, 2010.

[61] Laible, G., Spratt, B. G., and Hakenbeck, R., "Interspecies recombinational events during the evolution of altered pbp 2x genes in penicillin-resistant clinical isolates of streptococcus pneumoniae," *Molecular Microbiology*, vol. 5, no. 8, pp. 1993–2002, 1991.

[62] Lancefield, R. C., "A SEROLOGICAL DIFFERENTIATION OF HUMAN AND OTHER GROUPS OF HEMOLYTIC STREPTOCOCCI," *The Journal of Experimental Medicine*, vol. 57, no. 4, pp. 571–595, 1933.

[63] Langmead, B. and Salzberg, S. L., "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, 2012.

[64] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L., "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome.," *Genome biology*, vol. 10, no. 3, p. R25, 2009.

[65] Larsen, M. V., Cosentino, S., Rasmussen, S., Friis, C., Hasman, H., Marvig, R. L., Jelsbak, L., Sicheritz-Pontén, T., Ussery, D. W., Aarestrup, F. M., and Lund, O., "Multilocus Sequence Typing of Total Genome Sequenced Bacteria," *Journal of Clinical Microbiology*, vol. 50, no. 4, pp. 1355–1361, 2012.

[66] Lawrence, M. S., Stojanov, P., Polak, P., Kryukov, G. V., Cibulskis, K., Sivachenko, A., Carter, S. L., Stewart, C., Mermel, C. H., Roberts, S. a., Kiezun, A., Hammerman, P. S., McKenna, A., Drier, Y., Zou, L., Ramos, A. H., Pugh, T. J., Stransky, N., Helman, E., Kim, J., Sougnez, C., Ambrogio, L., Nickerson, E., Shefler, E., Cortés, M. L., Auclair, D., Saksena, G., Voet, D., Noble, M., DiCara, D., Lin, P., Lichtenstein, L., Heiman, D. I., Fennell, T., Imielinski, M., Hernandez, B., Hodis, E., Baca, S., Dulak, A. M., Lohr, J., Landau, D.-A., Wu, C. J., Melendez-Zajgla, J., Hidalgo-Miranda, A., Koren, A., McCarroll, S. a., Mora, J., Lee, R. S., Crompton, B., Onofrio, R., Parkin, M., Winckler, W., Ardlie, K., Gabriel, S. B., Roberts, C. W. M., Biegel, J. a., Stegmaier, K., Bass, A. J., Garraway, L. a., Meyerson, M., Golub, T. R., Gordenin, D. a., Sunyaev, S., Lander, E. S., and Getz, G., "Mutational heterogeneity in cancer and the search for new cancer-associated genes.," *Nature*, vol. 499, no. 7457, pp. 214–8, 2013.

[67] Li, G., Hu, F. Z., Yang, X., Cui, Y., Yang, J., Qu, F., Gao, G. F., and Zhang, J. R., "Complete genome sequence of Streptococcus pneumoniae Strain ST556, a Multidrug-Resistant Isolate from an Otitis Media Patient," *Journal of Bacteriology*, vol. 194, no. 12, pp. 3294–3295, 2012.

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

[69] Li, H. and Durbin, R., "Fast and accurate long-read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[70] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R., "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[71] Li, H. and Homer, N., "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[72] Li, H., Ruan, J., and Durbin, R., "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Research*, vol. 18, no. 11, pp. 1851–1858, 2008.

[73] Li, R., Li, Y., Kristiansen, K., and Wang, J., "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[74] Li, R., Yu, C., Li, Y., Lam, T. W., Yiu, S. M., Kristiansen, K., and Wang, J., "SOAP2: An improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[75] Li, Y., Hu, Y., Bolund, L., and Wang, J., "State of the art de novo assembly of human genomes from massively parallel sequencing data," *Human Genomics*, vol. 4, no. 4, pp. 271–277, 2010.

[76] Liu, B., Shi, Y., Yuan, J., Hu, X., Zhang, H., Li, N., Li, Z., Chen, Y., Mu, D., and Fan, W., "Estimation of genomic characteristics by analyzing k-mer frequency in de novo genome projects," *arXiv*, p. 1308.2012, 2013.

[77] Loman, N. J., Misra, R. V., Dallman, T. J., Constantinidou, C., Gharbia, S. E., Wain, J., and Pallen, M. J., "Performance comparison of benchtop high-throughput sequencing platforms," *Nature biotechnology*, vol. 30, no. 5, pp. 434–439, 2012.

[78] Maiden, M., Bygraves, J., Feil, E., Morelli, G., Russell, J., Urwin, R., Zhang, Q., Zhou, J., Zurth, K., Caugant, D., Feavers, I., Achtman, M., and Spratt, B., "Multilocus sequence typing: a portable approach to the identification of clones within populations of pathogenic microorganisms," *Proceedings of the National Academy of Sciences*, vol. 95, no. 6, pp. 3140–5, 1998.

[79] Maiden, M. C. J., "Multilocus Sequence Typing of Bacteria," *Annual Review of Microbiology*, vol. 60, pp. 561–588, 2006.

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

[81] Mardis, E. R., "A decade's perspective on DNA sequencing technology," *Nature*, vol. 470, pp. 198–203, feb 2011.

[82] Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., Dewell, S. B., Du, L., Fierro, J. M., Gomes, X. V., Godwin, B. C., He, W., Helgesen, S., Ho, C. H., Ho, C. H., Irzyk, G. P., Jando, S. C., Alenquer, M. L. I., Jarvie, T. P., Jirage, K. B., Kim, J.-B., Knight, J. R., Lanza, J. R., Leamon, J. H., Lefkowitz, S. M., Lei, M., Li, J., Lohman, K. L., Lu, H., Makhijani, V. B., McDade, K. E., McKenna, M. P., Myers, E. W., Nickerson, E., Nobile, J. R., Plant, R., Puc, B. P., Ronan, M. T., Roth, G. T., Sarkis, G. J., Simons, J. F., Simpson, J. W., Srinivasan, M., Tartaro, K. R., Tomasz, A., Vogt, K. A., Volkmer, G. A., Wang, S. H., Wang, Y., Weiner, M. P., Yu, P., Begley, R. F., and Rothberg, J. M., "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–80, 2005.

[83] Martin, C., Sibold, C., and Hakenbeck, R., "Relatedness of penicillin-binding protein 1a genes from different clones of penicillin-resistant Streptococcus pneumoniae isolated in South Africa and Spain," *The EMBO Journal*, vol. 11, pp. 3831–6, nov 1992.

[84] Marttila, H. J., Soini, H., Eerola, E., Vyshnevskaya, E., Vyshnevskiy, B. I., Otten, T. F., Vasilyef, A. V., and Viljanen, M. K., "A Ser315Thr Substitution in KatG is Predominant in Genetically Heterogeneous Multidrug-Resistant Mycobacterium tuberculosis isolates Originating from the St. Petersburg Area in Russia," *Antimicrobial Agents and Chemotherapy*, vol. 42, no. 9, pp. 2443–2445, 1998.

[85] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., and DePristo, M. A., "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297–1303, 2010.

[86] Melsted, P. and Pritchard, J. K., "Efficient counting of k-mers in DNA sequences using a bloom filter.," *BMC bioinformatics*, vol. 12, no. 1, p. 333, 2011.

[87] Metzker, M. L., "Sequencing technologies - the next generation.," *Nature reviews. Genetics*, vol. 11, pp. 31–46, jan 2010.

[88] Mokorousov, I., Narvskaya, O., Otten, T., Limeschenko, E., Steklova, L., and Vyshnevskiy, B., "High Prevalence of katG Ser315Thr Substitution Among IsoniazidResistant Mycobacterium tuberculosis Clinical Isolates from Northwestern Russia, 1996 to 2001," *Antimicrobial Agents and Chemotherapy*, vol. 46, no. 5, pp. 1417–1424, 2002.

[89] Moore, G. E., "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[90] Musser, J. M., Kapur, V., Williams, D. L., Kreiswirth, B. N., van Soolingen, D., and van Embden, J. D. A., "Characterization of the Catalase-Peroxidase gene (katG) and inhA Locus in Isoniazid-resistant and -Susceptible Strains of Mycobacterium tuberculosis by Automated DNA Sequencing: Restricted Array of Mutations Associated with Drug Resistance," *The Journal of infectious diseases*, vol. 173, pp. 196–202, 1996.

[91] Newman, A. M., Bratman, S. V., To, J., Wynne, J. F., Eclov, N. C. W., Modlin, L. A., Liu, C. L., Neal, J. W., Wakelee, H. A., Merritt, R. E., Shrager, J. B., Loo, B. W., Alizadeh, A. A., and Diehn, M., "An ultrasensitive method for quantitating circulating tumor DNA with broad patient coverage," *Nature Medicine*, vol. 20, no. 5, pp. 548–554, 2014.

[92] Nielsen, R., Paul, J. S., Albrechtsen, A., and Song, Y. S., "Genotype and SNP calling from next-generation sequencing data," *Nature Reviews*, vol. 12, no. 6, pp. 443–451, 2011.

[93] O'Brien, K. L., Wolfson, L. J., Watt, J. P., Henkle, E., Deloria-Knoll, M., McCall, N., Lee, E., Mulholland, K., Levine, O. S., and Cherian, T., "Burden of disease caused by Streptococcus pneumoniae in children younger than 5 years: global estimates," *The Lancet*, vol. 374, no. 9693, pp. 893–902, 2009.

[94] Olson, N. D., Lund, S. P., Colman, R. E., Foster, J. T., Sahl, J. W., Schupp, J. M., Keim, P., Morrow, J. B., Salit, M. L., and Zook, J. M., "Best practices for evaluating single nucleotide variant calling methods for microbial genomics," *Frontiers in Genetics*, vol. 6, no. JUL, pp. 1–15, 2015.

[95] Organization, W. H., "The end tb strategy," 2016. Accessed June 02, 2016.

[96] Pabinger, S., Dander, A., Fischer, M., Snajder, R., Sperk, M., Efremova, M., Krabichler, B., Speicher, M. R., Zschocke, J., and Trajanoski, Z., "A survey of tools for variant analysis of next-generation genome sequencing data," *Briefings in Bioinformatics*, vol. 15, no. 2, pp. 256–278, 2014.

[97] Parsons, D. W., Wang, T.-L., Samuels, Y., Bardelli, A., Cummins, J. M., DeLong, L., Silliman, N., Ptak, J., Szabo, S., Willson, J. K. V., Markowitz, S., Kinzler, K. W., Vogelstein, B., Lengauer, C., and Velculescu, V. E., "Colorectal cancer: mutations in a signalling pathway.," *Nature*, vol. 436, no. August, p. 792, 2005.

[98] Patro, R., Mount, S. M., and Kingsford, C., "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms.," *Nature biotechnology*, vol. 32, pp. 462–4, may 2014.

[99] Prahallad, A., Sun, C., Huang, S., Di Nicolantonio, F., Salazar, R., Zecchin, D., Beijersbergen, R. L., Bardelli, A., and Bernards,

R., "Unresponsiveness of colon cancer to BRAF(V600E) inhibition through feedback activation of EGFR," *Nature*, vol. 483, no. 7388, pp. 100–103, 2012.

[100] Reuter, S., Ellington, M. J., Cartwright, E. J. P., Köser, C. U., Török, M. E., Gouliouris, T., Harris, S. R., Brown, N. M., Holden, M. T. G., Quail, M., Parkhill, J., Smith, G. P., Bentley, S. D., and Peacock, S. J., "Rapid Bacterial Whole-Genome Sequencing to Enhance Diagnostic and Public Health Microbiology," *JAMA Internal Medicine*, vol. 173, no. 15, pp. 1397–404, 2013.

[101] Rimmer, A., Phan, H., Mathieson, I., Iqbal, Z., Twigg, S. R. F., Wilkie, A. O. M., McVean, G., and Lunter, G., "Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications," *Nature Genetics*, vol. 46, no. 8, pp. 912–918, 2014.

[102] Rizk, G., Lavenier, D., and Chikhi, R., "DSK: k-mer counting with very low memory usage.," *Bioinformatics (Oxford, England)*, vol. 29, pp. 652–653, mar 2013.

[103] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., and Yorke, J. A., "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, 2004.

[104] Rouse, D. a., Li, Z., Bai, G. H., Morris, S. L., Rouse, D. a., Li, Z., Bai, G.-h., and Morris, S. L., "Characterization of the katG and inhA Genes of Isoniazid-Resistant Clinical Isolates of Mycobacterium tuberculosis," *Antimicrobial Agents and Chemotherapy*, vol. 39, no. 11, pp. 2472–2477, 1995.

[105] Sanger F, C. A., "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase," *Journal of Molecular Biology*, vol. 94, no. 3, pp. 441–446, 1975.

[106] Saunders, N. J., Trivedi, U. H., Thomson, M. L., Doig, C., Laurenson, I. F., and Blaxter, M. L., "Deep resequencing of serial sputum isolates of Mycobacterium tuberculosis during therapeutic failure due to poor compliance reveals stepwise mutation of key resistance genes on an otherwise stable genetic background," *Journal of Infection*, vol. 62, no. 3, pp. 212–217, 2011.

[107] Sboner, A., Mu, X. J., Greenbaum, D., Auerbach, R. K., and Gerstein, M. B., "The real cost of sequencing: higher than you think!," *Genome Biology*, vol. 12, no. 8, p. 125, 2011.

[108] Schürch, A. C. and van Soolingen, D., "DNA fingerprinting of Mycobacterium tuberculosis: from phage typing to whole-genome sequencing," *Infection, Genetics and Evolution*, vol. 12, pp. 602–9, jun 2012.

[109] Siu, G. K. H., Zhang, Y., Lau, T. C. K., Lau, R. W. T., Ho, P.-L., Yew, W.-W., Tsui, S. K. W., Cheng, V. C. C., Yuen, K.-Y., and

Yam, W.-C., "Mutations outside the rifampicin resistance-determining region associated with rifampicin resistance in Mycobacterium tuberculosis," *Journal of Antimicrobial Chemotherapy*, vol. 66, no. 4, pp. 730–733, 2011.

[110] Smith, L. M., Sanders, J. Z., Kaiser, R. J., Hughes, P., Dodd, C., Connell, C. R., Heiner, C., Kent, S. B. H., and Hood, L. E., "Fluorescence detection in automated DNA sequence analysis," *Nature*, vol. 321, pp. 674–679, jun 1986.

[111] Smith, T. and Waterman, M., "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, mar 1981.

[112] Stodden, V. and Miguez, S., "Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research," *Journal of Open Research Software*, vol. 2, no. 1, p. 21, 2014.

[113] Torres, J. N., Paul, L. V., Rodwell, T. C., Victor, T. C., Amallraja, A. M., Elghraoui, A., Goodmanson, A. P., Ramirez-Busby, S. M., Chawla, A., Zadorozhny, V., Streicher, E. M., Sirgel, F. a., Catanzaro, D., Rodrigues, C., Gler, M. T., Crudu, V., Catanzaro, A., and Valafar, F., "Novel katG mutations causing isoniazid resistance in clinical M. tuberculosis isolates," *Emerging Microbes and Infections*, vol. 4, no. April, p. e42, 2015.

[114] van Belkum, A., Tassios, P., Dijkshoorn, L., Haeggman, S., Cookson, B., Fry, N., Fussing, V., Green, J., Feil, E., Gerner-Smidt, P., Brisse, S., and Struelens, M., "Guidelines for the validation and application of typing methods for use in bacterial epidemiology," *Clinical Microbiology and Infection*, vol. 13, no. 3, pp. 1–46, 2007.

[115] Vasconcellos, S. E. G., Acosta, C. C., Gomes, L. L., Conceição, E. C., Lima, K. V., de Araujo, M. I., Leite, M. D. L., Tannure, F., Caldas, P. C. D. S., Gomes, H. M., Santos, A. R., Gomgnimbou, M. K., Sola, C., Couvin, D., Rastogi, N., Boechat, N., and Suffys, P. N., "Strain Classification of Mycobacterium tuberculosis Isolates in Brazil Based on Genotypes Obtained by Spoligotyping, Mycobacterial Interspersed Repetitive Unit Typing and the Presence of Large Sequence and Single Nucleotide Polymorphism," *PLoS ONE*, vol. 9, no. 10, p. e107747, 2014.

[116] Votintseva, A. A., Pankhurst, L. J., Anson, L. W., Morgan, M. R., Gascoyne-Binzi, D., Walker, T. M., Quan, T. P., Wyllie, D. H., Del Ojo Elias, C., Wilcox, M., Walker, A. S., Peto, T. E. A., and Crook, D. W., "Mycobacterial DNA Extraction for Whole-Genome Sequencing from Early Positive Liquid (MGIT) Cultures," *Journal of Clinical Microbiology*, vol. 53, pp. 1137–1143, apr 2015.

[117] Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K. D., Mitchell, I. M.,

Plumbley, M. D., Waugh, B., White, E. P., and Wilson, P., “Best Practices for Scientific Computing,” *PLoS biology*, vol. 12, p. e1001745, jan 2014.

[118] Wood, D. E. and Salzberg, S. L., “Kraken: ultrafast metagenomic sequence classification using exact alignments,” *Genome Biology*, vol. 15, pp. 1–12, jan 2014.

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

[120] Zerbino, D. R. and Birney, E., “Velvet: Algorithms for de novo short read assembly using de Bruijn graphs,” *Genome Research*, vol. 18, no. 5, pp. 821–829, 2008.

[121] Zhang, Q., Pell, J., Canino-koning, R., Howe, A. C., Brown, C. T., Lansing, E., Genetics, M., and Sciences, M., “These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure,” *arXiv*, 2014.

VITA

Peter Audano was born in Fort Myers FL and grew up in Gainesville FL. He finished high school at a local community college where he got his early education in computer programming. After high school, he moved to Atlanta and began working for Earthlink Inc. in 2001. In 2002, he enrolled in the computer science program at Southern Polytechnic State University (SPSU; now part of Kennesaw University). After three years, he left Earthlink to work for Internet Security Systems (ISS) where he led a product support team before moving to quality assurance. There, he designed and built a modular test automation system for network security devices. IBM acquired ISS in 2006, but Pete's role did not change. In 2008, he graduated with his BS in Computer Science and continued to work at ISS. In 2012, he left IBM to pursue a graduate degree in bioinformatics at the Georgia Institute of Technology.