Scalable methods for storage, processing and analysis of sequencing datasets

by

Deniz Yorukoglu

B.S., Sabanci University (2009)
M.S., Simon Fraser University (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2017

©2017 Massachusetts Institute of Technology. All rights reserved.

Signature redacted

Signature of Author: ..................................
Department of Electrical Engineering and Computer Science
January 31, 2017

Certified by: Signature redacted..........................
Bonnie Berger
Simons Professor of Mathematics and Professor of Computer Science
Thesis Supervisor

Signature redacted

Accepted by: ...........................................
Professor Leslie A. Kolodziejski
Chair of the Committee on Graduate Students
Scalable methods for storage, processing and analysis of sequencing datasets

by

Deniz Yorukoglu

Submitted to the Department of Electrical Engineering and Computer Science on January 31, 2017, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Massive amounts of next-generation sequencing (NGS) reads generated from sequencing machines around the world have revolutionized biotechnology enabling wide-scale disease and variation studies, personalized medicine and helping us understand our evolutionary history. However, the amount of sequencing data generated every day increases at an exponential rate posing an imminent need for smart algorithmic solutions to handle massive sequencing datasets and efficiently extract the useful knowledge within them. This thesis consists of four research contributions on these two fronts. First, we present a computational framework that leverages the redundancy within large genomic datasets for performing faster read-mapping while improving sensitivity. Second, we describe a lossy compression method for quality scores within sequencing datasets that strikingly improves the downstream accuracy for genotyping. Third, we introduce a Bayesian framework for accurate diploid and polyploid haplotype reconstruction of an individual genome using NGS datasets. Lastly, we extend this haplotype reconstruction framework to high-throughput transcriptome sequencing datasets.

Thesis Supervisor: Bonnie Berger
Title: Simons Professor of Mathematics and Professor of Computer Science
Acknowledgments

I would like to thank my supervisor Bonnie Berger for her utmost support throughout my PhD studies and broadening my horizons with regards to what is possible with algorithms for genomics. I owe enormous gratitude to her for her guidance throughout my PhD studies. I would also like to thank my thesis readers, Eric Alm and Samuel Madden for their constructive comments and helpful suggestions on my thesis work.

The work included in this thesis would not be possible without Yun William Yu, Emily Berger and Jian Peng; it was an immense joy to work and learn with them and push through challenges together with teamwork.

I would also like to thank all Berger Lab members and alumni who have shaped my perspective on science and research throughout my PhD studies, especially Po-Ru Loh, George Tucker, Mark Lipson, Noah Daniels, Sean Simmons, Alex Levin, Luke Hutchison, Bianca Dumitrascu, Leonid Chindelevitch, Maria Rodriguez, Hoon Cho, Sumaiya Nazeen, Hilary Finucane, Ariya Shajii, Yaron Orenstein, Perry Palmedo, Tristan Bepler, Michael Baym, and Manolis Kellis. As well as many external collaborators that I have learned from, especially Isaac Kohane, Oliver Cooper, David Gifford, Abhishek Sarkar, Laxmi Parida, Saugata Basu, Filippo Utro, William Brockman, Samuel Gross, Evan Senter, Michelle Dimon, Alper Kucukural as well as my former advisors Cenk Sahinalp and Ugur Sezerman. I also owe deep gratitude to Patrice Macaluso, Janet Fischer, and MIT ISO staff for their continued kindness and administrative support.

Throughout my PhD, I was generously supported by fellowships from Akamai, IBM, HHMI and research funding from NIH.

Last but not the least, I would like thank my family and friends for their invaluable support and patience throughout these years of my life that resulted in this work. I would like to dedicate this thesis to my nephew, Eren Sungur. I sincerely hope that one day; if he wishes to pursue a scientific career, my studies can serve as a good example.
Previous Publications of This Work

Results described in Chapter 2 were published in Nature Biotechnology 34, 374-376, with co-authors Yun William Yu, Jian Peng and Bonnie Berger.

Results described in Chapter 3 were initially published in RECOMB 2014 with co-authors Yun William Yu and Bonnie Berger. Later a more powerful method and more comprehensive analysis results were published in Nature Biotechnology 33, 240-243 with co-authors Yun William Yu, Jian Peng and Bonnie Berger.

Results described in Chapter 4 were published in RECOMB 2014 and PLOS Computational Biology 10(3):e1003502 with co-authors Emily Berger, Jian Peng and Bonnie Berger.

Results described in Chapter 5 were published in RECOMB 2015 with co-authors Emily Berger and Bonnie Berger.
5.3 Results .................................................. 165
  5.3.1 Theoretical performance .......................... 165
  5.3.2 Experimental results .............................. 166

6 Conclusion and Discussion .............................. 171
  6.1 Compression ......................................... 171
  6.2 Compressive acceleration ............................ 173
  6.3 Haplotype reconstruction ............................ 176
List of Figures

1-1 Cost of DNA sequencing in the past decade and a half. .................. 19
1-2 Sequencing by synthesis. ......................................................... 22
2-1 Runtime and sensitivity comparison results between conventional read mapping methods and CORA. .................................................. 33
2-2 Conventional read mapping methods and compressively accelerated read mapping framework. ......................................................... 37
2-3 Compact representation of homology table with exact and inexact homologies. ................................................................. 40
2-4 Seed position selection for inexact homology table construction. .... 45
2-5 Estimation of redundancy within read datasets in absence of sequencing error. ................................................................. 57
2-6 Sublinearity analysis of CORA framework on simulated paired-end read data. ................................................................. 59
2-7 Homology table traversal scheme of CORA framework. ............... 62
2-8 An overview of CORA framework and input/output relations between different stages of the CORA software .................................. 78
2-9 Runtime and sensitivity comparison results for whole genome ungapped and gapped best-mapping. ................................................................. 85
3-1 Visualization of Quartz algorithm. .................................................. 108
3-2 Downstream SNP calling before and after Quartz compression ....... 121
3-3 Downstream indel calling before and after Quartz compression. .... 122
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>Changes in ROC curves when quality score values are removed or assigned different default values within the Quartz algorithm.</td>
<td>123</td>
</tr>
<tr>
<td>3-5</td>
<td>Quartz compared to other methods on NA12878 dataset.</td>
<td>124</td>
</tr>
<tr>
<td>3-6</td>
<td>Quartz compared to other methods on NA19240 dataset.</td>
<td>126</td>
</tr>
<tr>
<td>3-7</td>
<td>ROC curves for low-coverage datasets.</td>
<td>127</td>
</tr>
<tr>
<td>3-8</td>
<td>Quartz compared to other methods on E. coli dataset</td>
<td>129</td>
</tr>
<tr>
<td>4-1</td>
<td>Loss of function in a sample pentaploid genome.</td>
<td>132</td>
</tr>
<tr>
<td>4-2</td>
<td>Proportion of perfectly phased SNP pairs and vector error rate.</td>
<td>148</td>
</tr>
<tr>
<td>4-3</td>
<td>Performance of HapTree and HapCompass on triploid genomes.</td>
<td>149</td>
</tr>
<tr>
<td>4-4</td>
<td>Performance of HapTree and HapCompass on tetraploid genomes.</td>
<td>149</td>
</tr>
<tr>
<td>4-5</td>
<td>HapTree performance over varied error rates and coverages on simulated triploid genomes</td>
<td>150</td>
</tr>
<tr>
<td>5-1</td>
<td>A toy example demonstrating haplotype phasing capabilities of various approaches</td>
<td>156</td>
</tr>
<tr>
<td>5-2</td>
<td>Likelihood of concordant expression as a function of coverage and differential haplotypic expression</td>
<td>165</td>
</tr>
<tr>
<td>5-3</td>
<td>Coverage needed to obtain high likelihood of concordant expression</td>
<td>166</td>
</tr>
<tr>
<td>5-4</td>
<td>Rate of concordantly expressed SNPs and number of SNPs phased as a function of $\lambda$.</td>
<td>169</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Computational cost of CORA's homology table construction stage.</td>
<td>48</td>
</tr>
<tr>
<td>2.2</td>
<td>Computational cost of CORA's collapsing stage.</td>
<td>60</td>
</tr>
<tr>
<td>2.3</td>
<td>Computational cost of CORA's homology table traversal stage.</td>
<td>67</td>
</tr>
<tr>
<td>2.4</td>
<td>All-mapping and coarse mapping runtimes for different Hamming distances.</td>
<td>86</td>
</tr>
<tr>
<td>2.5</td>
<td>All-mapping sensitivity results for different Hamming distances.</td>
<td>87</td>
</tr>
<tr>
<td>2.6</td>
<td>Runtime and sensitivity results of all-mapping when mapping recovery is disabled.</td>
<td>87</td>
</tr>
<tr>
<td>3.1</td>
<td>Compression benchmarks on NA12878.</td>
<td>118</td>
</tr>
<tr>
<td>3.2</td>
<td>Runtime and compression benchmarks for different k-mer lengths and Hamming distance.</td>
<td>118</td>
</tr>
<tr>
<td>3.3</td>
<td>Runtime benchmarks for Bowtie2 + GATK and BWA + SAMtools pipelines.</td>
<td>119</td>
</tr>
<tr>
<td>3.4</td>
<td>ROC AUC values for original uncompressed vs Quartz compressed read datasets.</td>
<td>125</td>
</tr>
<tr>
<td>3.5</td>
<td>Time and storage requirements after Quartz compression and GATK recalibration</td>
<td>129</td>
</tr>
<tr>
<td>4.1</td>
<td>Whole-genome phasing results of HapTree and HapCUT using 454 and Illumina sequencing datasets.</td>
<td>151</td>
</tr>
<tr>
<td>5.1</td>
<td>Distribution of number of heterozygous-SNPs overlapped by each read in GM12878 RNA-seq data.</td>
<td>167</td>
</tr>
</tbody>
</table>
5.2 Haplotype reconstruction results from HapTree-X and HapCut using whole-genome sequencing and RNA-seq datasets
Chapter 1

Introduction

1.1 Computational genomics in the era of next-generation sequencing

In no other biological problem domain are we currently running up against the limits of data storage and processing as much as we are with the challenges of next-generation sequencing (NGS) read data. Massive amounts of NGS reads generated from sequencing machines around the world have revolutionized biotechnology enabling wide-scale disease and variation studies, personalized medicine and helping us understand our evolutionary history. However, the amount of sequencing data generated every day increases at a faster exponential rate than both computing power and storage per dollar.

Two of the key challenges in the big data era of genomics are how to handle massive datasets in an efficient manner and how to make sense of the data at large scales. Tackling these challenges requires both efficient algorithms coupled with compact data structures and intelligent frameworks that can effectively mine the useful knowledge within large biological datasets. This thesis highlights four studies that contribute to the field on either front.
1.1.1 Genomes and next-generation sequencing

DNA, or deoxyribonucleic acid, is a long string of nucleotides that contain one of four types of chemical bases: adenine, cytosine, guanine and thymine (shortly referred as A, C, G, and T). These nucleotides pair up with each other forming base pairs (A with T and G with C) and together form a double-stranded DNA sequence, which contains a comprehensive set of genetic instructions within each cell that determine many characteristics defining an organism and also form the main hereditary component. For eukaryotic species, including humans, these double-stranded DNA molecules are packaged into compact structures within the nucleus called chromosomes, forming the genome of the organism.

The Human Genome Project, a major scientific undertaking that cost 2.7 billion USD FY 1991 (4.8 billion USD adjusted for FY 2016), took 13 years to complete and produced a high-quality reference of the whole human genome constructed from several individuals by 2003. This fully assembled human reference genome consists of approximately 3 billion base-pairs representing a haploid human genome (or about half of the number of base-pairs in a diploid human cell). Since then, there has been an exponential decrease in terms of the cost of sequencing a human genome (Figure 1-1); dropping to a million dollars per genome (454 Life Sciences) in 2007, reducing to around $50,000 before the end of the decade (Helicos & Illumina) and dropping to less than $1000 per genome in 2014 (Illumina). The majority of the work presented in this thesis is focused on sequencing datasets generated from the recent iterations of these sequencing technologies (dubbed next-generation sequencing technologies), particularly datasets produced from Illumina sequencing technologies, most common type of sequencing data available as of 2016.

While next-generation sequencing (NGS) technologies provide a cheap solution for analyzing an individual’s genome, the data produced from them has certain properties that give rise to interesting and non-trivial computational problems. One such property is that, DNA sequencers cannot read a whole chromosome at once and instead long DNA sequences need to be chopped into much shorter fragments of about

1Also including sex chromosomes X and Y and the mitochondrial DNA.
Figure 1-1: Cost of DNA sequencing in the past decade and a half in terms of sequencing a human genome (left panel) and the raw megabase of DNA sequence (right panel). Plots indicate that sequencing costs have dropped drastically since 2008 with the arrival of next-generation sequencing technologies (Image source: Wetterstrand KA. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP) Available at: www.genome.gov/sequencingcostsdata. Accessed 12/13/2016).

300-700 base pairs. For each fragment, around 75 to 150 bases are read from both ends of the fragment, producing paired-end reads. Therefore one of the most critical stages of genomics analysis pipelines is to figure out where these reads are originating from within the genome of the sequenced sample (i.e. which position of which chromosome). One way to perform this is through the use of a reference genome by aligning the reads to regions of the reference genome with high sequence similarity, a task commonly referred as read-mapping. A wide variety of efficient algorithms and data structures have been devised for this particular problem in the past decade, yet it still remains as one of the most computationally intensive components of NGS processing pipelines.

Another interesting property of sequencing datasets is that, each sequenced base in a read gets assigned a quality score that represents the confidence of the sequencing machine for calling that particular base instead of the other bases. While the alphabet size of read sequences is only 5 (A, C, G, T, corresponding to the four nucleotide bases, and N for an undetermined base), quality scores can take a much larger range of values (usually ~ 40), making it a bigger challenge to store them efficiently. That is why finding efficient ways to store quality scores is as important as storing the sequences themselves, leading to additional computational challenges to be solved for efficient
storage of NGS data.

Once a read dataset is mapped to the reference genome, it needs to be further analyzed in order to discover genetic variants (nucleotides or a sequence of nucleotides that differ in the individual than the reference genome). This process of determining the differences in the genome of an individual, and delineating the variant alleles present in the individual is called genotyping. While genotyping provides a clear picture of mutations in the individual that may result in a phenotype, it does not differentiate between whether multiple mutations exist on the same (paternal or maternal) copy of a chromosome or on different copies. One of the important problems to be solved in NGS analysis is to find out which alleles are inherited together from a single parent and how these affect different phenotypes. Similar to processing and storing NGS datasets, extracting such useful biological information from these datasets also require efficient and effective analysis approaches to be developed.

1.1.2 Challenges of sequencing data deluge

Advances in next-generation sequencing (NGS) technologies are resulting in a deluge of genomic information, with the rate of sequencing read datasets generated each year outpacing even Moore’s and Kryder’s laws in terms of the exponential increase in transistor density and hard drive capacity. By one estimate, the projection of genomic data growth to the year 2025 is 1 zetta-bases (i.e. 10^{21} bases) per year, estimated to cost 2-40 exabytes to store and 2 trillion CPU hours to process [1]. Such a projection indicates more demanding storage and processing costs for genomics than astronomy and online video hosting (e.g. YouTube) combined by 2025.

While from a data distribution perspective, total amount bandwidth needed for genomic data transfer is not as substantial as large video hosting services, the added concern for privacy due to medically sensitive information makes this task significantly more challenging; a large-scale genomic data leak will result in a range of issues for the leaked individuals and their relatives such as, revealing non-paternity and planting synthetic DNA of leaked individuals at crime scenes [2], as well as denied coverage by health/life insurance companies to those with genetic mutations linked
to disease-risk. Once leaked, such information will be virtually impossible to contain and the harm done will be difficult to amend. Such concerns make biomedical institutions and hospitals less eager to delegate large-scale storage and computing needs to cloud computing solutions without hard privacy guarantees, further emphasizing the importance for finding secure and standardized privacy solutions as well as efficient algorithms and data structures to increase the capacity of genomic data we can store and process on a private cloud in the mean time. Ultimately, these cost effective solutions will also directly benefit cloud-based storage and processing.

**Keeping ‘raw’ data around for bookkeeping**

In the NGS era of genomics, the ‘raw’ sequencing read datasets (i.e. text files made up of read sequences and quality scores) form the bulk of the data that need to be stored and processed for most genomic analyses. These raw reads are often stored for bookkeeping purposes and represent the most upstream information commonly available for NGS analysis studies. As the amount of sequencing data generated per dollar increases at a bigger exponential rate than the increase in the amount of storage available for dollar, it is becoming increasingly impractical to indefinitely store raw sequencing data for later processing in an uncompressed state.

While sequencing centers, repositories and researchers generally compress their sequencing datasets with lossless compression methods (e.g. gzip, fastqz [3], SRA [4]) in order to reduce storage costs, they are often less eager to employ lossy compression techniques to further reduce storage cost. This is usually in the interest of keeping the data intact in the ‘raw’ format with the concern that any divergence from the ‘raw’ data will lead to loss in information.

However, sequencing read datasets are not the most raw data form produced by sequencing machines in the first place. The most common sequencing technology currently available (i.e. Illumina’s sequencing by synthesis technology) operate by means of adding fluorescently labeled nucleotides onto a solid transparent surface (i.e. a flow cell) that holds the single-stranded DNA fragments waiting to be sequenced. At each iterative cycle, one fluorescently labelled nucleotide attaches to each fragment
(to the first unpaired base) emitting a frequency of light that is captured by a camera (Figure 1-2). The resulting images are then processed by a base-calling software and converted into the read sequences and the quality scores that we see in conventional sequencing datasets.

**Figure 1-2:** During sequencing by synthesis, the forward strand of the fragment (left panel) is attached to the flow cell surface and a double stranded primer is appended to it. The incomplete primer (left panel, blue dots) is extended by fluorescently-tagged nucleotides, each addition corresponding to a sequencing cycle. A camera within the sequencer captures the fluorescent signal emitted at each cycle corresponding to the nucleotide added to the fragment (right panel, indicated by white circles) as well as many other fragments at different regions of the flowcell in parallel. These images are then processed through a base-calling software that outputs the read sequence as well as a quality score corresponding to each base of the read. (Image source: Courtesy of Illumina Inc.).

This process of converting image data into sequence and quality score information involves multiple non-trivial components, such as detecting the bright spots corresponding to nucleotides and distinguishing them both from each other and the background noise, extracting light intensities corresponding to each fragment and normalizing them by the relative intensity differences of four bases: A, C, G, T. Furthermore, the quality scores assigned to the base-calling confidence has its own caveats and biases, such as relying on precomputed quality tables for certain sequencing models which may differ from the sequencing data at hand. This non-trivial process that
converts images to read sequences and quality scores constitutes a ‘lossy’ transformation in itself, yet the sequencing images are rarely included in sequence repositories as the ‘raw’ form of the data for bookkeeping.

In this perspective, trying to store the read datasets completely intact will not be the right approach for tackling the sequencing data deluge in the upcoming years. It is not the raw reads and quality scores that are important for genomic analysis, it is the biological knowledge they provide about the sequenced individual. If the cost of storing a dataset can be greatly reduced while keeping the entirety of the useful information contained within intact, a method performing such a lossy transformation should be preferable for bookkeeping than storing all of the information as is. And in the case that a method provides a lossy transformation that demonstrates improvements in both storage as well as the usefulness of the transformed data in downstream uses, then such an approach should be easier to adopt for the analysis as well as for book-keeping purposes. Chapter 3 of this thesis describes such a compression method for quality scores.

More and more computation, but more of the same

As the amount sequencing data generated per dollar increases at a faster exponential rate than Moore’s law, the cost of processing the generated genomic data is becoming increasingly more challenging. However, for large sequencing datasets within the same species, each additionally sequenced individual has an increased likelihood of having reads (or long substrings of reads, e.g. mid-sized k-mers) that identically (or with very high similarity) match to one of the reads in the previously sequenced individuals. Therefore each sequenced individual adds less novel sequences and lowers the complexity per read for the overall corpus of reads.

While read datasets get more and more redundant with each added read dataset, the computational cost of most existing sequence alignment methods scale linearly (or very close to linearly) with the size of the input read dataset, hence the cost of aligning and analyzing NGS read data generated effectively increases exponentially every year as well, making it increasingly difficult to cope with the computational
demands of genomic analysis. Even today, computational analysis of genomic data is the main bottleneck of biomedical genomics studies, and not the data generation. Therefore it is important to build methods that can leverage the redundancy caused by the increase in data size for faster computation.

There has been some progress for utilizing the redundancy within large NGS datasets for faster read mapping, mainly through utilizing exact homology across reads by hashing full read sequences [5] or inexact homology via clustering [6], with the caveat of limited speedup for longer reads (>100bp) in the former and considerable loss in sensitivity in the latter. In Chapter 2 we present a different $k$-mer based approach that is able to provide both substantial acceleration and minimal or no loss in sensitivity.

In addition to the size of the read dataset, another source of redundancy is the reference genome itself. More than two thirds of the haploid human genome is formed of repetitive DNA [7] (a DNA sequence that appears at multiple places in the genome with high similarity) and this ratio goes up to 85% for some plant genomes (e.g. maize [8]).

For most read-mapping methods, the repetitiveness of the genome adversely affects the speed of mapping, due to substantially larger amount of similarity computations needed. A read that is sequenced from a unique region of the reference genome is often easy to locate; however, a read that is sequenced from a repetitive region of the genome would often require additional similarity computations in order to assess the likelihood of the read being sampled from each similar repeat region. A further problem is that these type of reads that require more resources to match to the reference appear at a higher frequency in the read datasets, simply due to higher coverage of a repeat DNA in the input dataset than DNA coming from a unique genomic region. For this reason, there is a quadratic relationship between the cost of mapping and the rate of repetitive sequences in the reference genome.

‘Compressive genomics’ (or ‘entropy-scaling search’) is a search paradigm that aims to alleviate this similarity computation explosion by means of exploiting the redundancy within genomic data [9, 10, 11]. The main approach they utilize is to
represent genomic data in a way that allows direct computation on the compressed data without decompression. This is achieved by separating the search problem into two components: coarse-search, where search is performed only on the cluster centers (representing the ‘compressed’ version of reference database) and fine-grained search where only the necessary parts of the genome is expanded and the alignment is performed as usual in this reduced search space. Through this methodology, Loh et al. [10], Daniels et al. [12] and Yu et al. [11] have developed compressively accelerated versions of the early generation of genomic search tools: BLAST [13], BLAT [14], BLASTP [15] and BLASTX [16]. The next-generation sequencing read-mapping method we describe in Chapter 2 employs a similar two-step ‘entropy-scaling search’ method for leveraging the redundancy of the reference genome, yet it is achieved through a homology table built on top of the reference genome instead of compressing the reference into cluster centroids.

Squeezing more information out of each read

While large-scale sequencing studies provide a lot of useful knowledge and insight into human variation [17], ancestral studies [18], population genomics [19], microbiome studies [20] and many other important problems in biomedicine, one of the pitfalls of such large-scale studies is that it is easy to keep generating ever-more data without making full use of the existing ones.

While generating more data produces more information and increases the statistical confidence of analysis by providing a more accurate estimate of the underlying distributions, the kind of the knowledge obtained stays the same after a certain level of coverage is reached. However, improving analyses by finding better ways to mine the existing data not only increases the amount of knowledge obtained but also provides a different perspective on the sample that is not necessarily available by merely sequencing more samples. One such example arises in the context of determining phase information across variants in an individual.

Diploid organisms, such as humans, obtain two copies of each chromosome from their parents, one maternal and one paternal. The task of haplotype phasing, also
referred to as haplotype reconstruction or haplotype assembly, aims to distinguish (or phase) maternally inherited genomic sequences in an individual from paternally derived ones [21]. There are a wide array of applications that benefit from haplotype phasing [22, 23], including understanding haplotype-specific effects in disease [24, 25, 26, 27, 28] and drug response [29], non-invasive fetal genotyping [30], and understanding evolutionary history [31, 32] and selection [33].

The use of sequence data for haplotype phasing enables distinguishing between the maternal and paternal alleles of the individual without relying on additional individuals and regardless of whether the mutation is novel or rare in the population. The main approach these methods use is to find reads that overlap multiple heterozygous mutations (positions where an individual has both a mutant allele and a wild-type allele), and phase alleles that appear together in a single-read. However, as next-generation sequencing reads are sampled from very short fragments of DNA (often <1000 bases), the range that they can phase through is very short. Whenever two consecutive heterozygous mutations on an individual chromosome are further apart than this range (which occurs very frequently), short genome sequencing reads will fail to create a phased haplotype that includes these two mutations.

One way to increase this phasing range is to use high-throughput transcriptome sequencing (RNA-seq) reads [34, 35, 36], which are produced by sequencing complementary DNA synthesized from RNA transcribed from genes [37, 38]. As genes often contain long introns that are spliced-out [39], transcriptome reads produced from them can span much longer ranges when mapped to the reference genome. This allows phasing mutations that are much further away than it is possible to phase with DNA sequencing, as long as the mutations are within the same gene. However, even with RNA-seq reads, the range is still limited by the transcript distance between the mutations. If there is a fully homozygous exonic region longer than the fragment length between two heterozygous mutations, there will not be any reads overlapping both mutations, hence it will not be possible to fully phase the gene.

The read-overlap information within RNA-seq datasets is not the only type of information useful for haplotype phasing; however, as the asymmetry of the distribu-
tion of RNA-seq reads between the two alleles of a mutation is also informative. One can use this differential allele specific expression (DASE) information within RNA-seq data to phase mutations within genes that are differentially expressed. This enables to use RNA-seq reads to phase mutations that are even further apart in the transcript than the maximum RNA-seq fragment length. By utilizing this DASE information on top of the read-overlap information, one can construct significantly more complete gene haplotypes.

This brings into question the efficacy of producing, storing and processing ever-more larger sequencing datasets to increase the power of genomic analyses, as opposed to also focusing on the quality of the analysis and finding ways to effectively use as much information as possible present within the datasets generated. In this particular case of RNA-seq read-based phasing, a higher coverage dataset still would not be able to provide reads that span two far-apart heterozygous mutations, yet mining DASE information within the existing set of RNA-seq reads might provide statistical power to confidently phase the mutations.

In Chapter 5 of this thesis, we present a haplotype phasing algorithm targeted for RNA-seq datasets that simultaneously utilizes read-overlap information and asymmetry of allele specific expression. This algorithm is built upon a Bayesian haplotype phasing framework for diploid and polyploid genomes, which is described in Chapter 4.

1.2 Organization of thesis

The remainder of this thesis is structured as four self-contained chapters describing separate research studies, and a final chapter that summarizes these research directions with a future outlook.

Chapter 2: Compressive Read Mapping for Next-Generation Sequencing

The high cost of mapping next-generation sequencing (NGS) reads onto a reference is a major bottleneck for sequencing analysis pipelines. This chapter introduces a
compressive read-mapping framework, CORA, that first maps reads to reads and reference to reference, exploiting inherent redundancies in both read and reference sequences, to accelerate read to reference mapping.

We use CORA to map paired-end reads from the 1000 Genomes Project to the human reference, eliminating redundant sequence comparisons and improving runtime by up to orders of magnitude with minimal or no loss in sensitivity, particularly for mapping multi-reads (reads that can be mapped to multiple loci in the reference genome). The relative speed advantage of our approach will increase with the explosion of NGS data and advances in sequencing technologies, allowing researchers to keep pace with this data onslaught.

Chapter 3: Scalable Quality Score Compression That Improves Accuracy

In this chapter, we describe a scalable quality score compression framework, Quartz, that substantially outperforms the compression ratio and speed of other de novo quality score compression methods while maintaining downstream genotyping accuracy.

Surprisingly, such a compression framework improves the SNP (i.e. single nucleotide polymorphism) genotyping accuracy on a gold-standard, real-life sequencing dataset (NA12878) using a k-mer density profile constructed from ~100 individuals from the 1000 Genomes Project. This improvement in downstream accuracy emerges from the observation that quality score values within NGS datasets are inherently encoded in the k-mer landscape of the genomic sequences.

Chapter 4: Polyploid Haplotype Reconstruction Using NGS Datasets

This chapter describes the polyploid haplotype reconstruction problem as well as a novel maximum-likelihood estimation solution for phasing mutations within diploid and polyploid genomes using high-throughput genome sequencing data.

While human and other eukaryotic genomes typically contain two copies of every chromosome, plants, yeast and fish such as salmon can have strictly more than two copies of each chromosome. Previous haplotype reconstruction studies have mainly focused on diploid genomes and are rarely scalable to genomes with higher ploidy, yet
computational investigations into polyploid genomes carry great importance, impacting plant, yeast and fish genomics, as well as the studies of the evolution of modern-day eukaryotes and (epi)genetic interactions between copies of genes. This necessitates efficient algorithms to obtain accurate and comprehensive phase information directly from the next-generation-sequencing read data in higher ploidy species.

We introduce an efficient statistical method, HapTree, for this task and show that our method significantly outperforms previous ones, in both accuracy and speed, for phasing triploid and higher ploidy genomes. Our method performs well on human diploid genomes as well, as demonstrated by our improved phasing of a real human genome sequencing dataset from 1000 Genomes Project.

Chapter 5: Haplotype Reconstruction of the Transcriptome

This chapter builds upon Chapter 4 and describes how to build a maximum-likelihood estimation solution for phasing genes haplotypes using high-throughput transcriptome sequencing (RNA-seq) datasets.

Previous haplotype reconstruction studies have ignored differential allele-specific expression in whole transcriptome sequencing (RNA-seq) data; however, intuition suggests that the asymmetry in this data (i.e. maternal and paternal haplotypes of a gene are differentially expressed) can be exploited to improve phasing power. In this chapter, we describe a novel integrative maximum-likelihood estimation framework, HapTree-X, for efficient, scalable haplotype assembly of an individual genome using transcriptomic and genomic NGS read datasets, making use of both read-overlap information and differential allele-specific expression.

We evaluate the performance of HapTree-X on real sequencing read data, both transcriptomic and genomic, from NA12878 (1000 Genomes Project and Gencode) and demonstrate that HapTree-X increases the number of SNPs that can be phased and sizes of phased-haplotype blocks, without compromising accuracy and improves the accuracy of phasing for the existing phased haplotype blocks.
Chapter 6: Conclusion and Discussion

This chapter contains an overall summary of the methods and results reported in the previous four chapters, as well as a discussion upon the changing landscape of computational genomics and the possible directions in which these studies can be improved and extended in order to deal with bigger challenges in genomics in the future.

We discuss the emerging long-read technologies (e.g. Pacific Biosciences and Oxford Nanopore), as well as linked-read sequencing technologies (e.g. 10X Genomics), and how these changes will affect the future genomics studies in terms of scalable storage, processing and analysis.
Chapter 2

Compressive Read Mapping for Next-Generation Sequencing

2.1 Overview

The analysis and storage of ever-increasing amounts of sequencing data present a huge computational challenge for the genomics community [9]. However, only a small proportion of this sequence information varies between individuals, and it is this variation that we hope to identify and understand. Recently, compressive genomics has been introduced as a way of improving the accuracy and efficiency of searching large sequencing databases [10]. Compressive genomics removes redundancies in genomic sequences and enables compressed data to be analyzed directly facilitating parsimonious storage and fast access. This approach has been shown to accelerate the performance of standard search tools, such as BLAST and BLAT [10]. Although some read-mapping tools also remove redundancies in the reference genome to facilitate mapping individual reads [40, 41, 42] they do not take full advantage of the redundancy across reads present in large sequencing datasets, which are often much larger and more redundant than the reference genome itself [43] (Figure 2-2).

This chapter describes a compressive read-mapping acceleration method (CORA), a computational framework that utilizes compression in order to boost the performance of existing read mappers [40, 41, 44] (Figure 2-2). CORA takes as input a
sequencing read dataset in FASTQ format and an off-the-shelf read-mapper. The read datasets are compressed into mid-size (typically ~30-60 bp) k-mer sets that contain only nonredundant sequence information. These representative k-mers are then mapped onto the reference genome using the existing read-mapper by means of a plug-in architecture. A high-resolution homology table is created for the reference sequence by mapping the reference to itself. The homology table contains all homologous pairs of loci in the reference above a similarity threshold, allowing fast, direct access to similar locations in the reference during mapping. The resulting mappings are outputted to a SAM file that can be readily integrated into existing sequence analysis pipelines.

CORA offers tractable mapping of terabyte-sized read datasets and achieves paired-end read mapping up to orders of magnitude faster than existing methods for mapping next-generation sequencing (NGS) reads from the 1000 Genomes Project and Mouse Genomes Project (Figures 2-1 and 2-9). Even for relatively small datasets (~100 GB and ~16× read depth-coverage), when such common mappers as BWA or Bowtie2 are plugged into CORA, our framework provides ~10 to ~4,700 times acceleration for 'all-mapping' (each read is mapped to all matching reference loci) with no loss in overall sensitivity, and ~2.7 to ~4.3 times for 'best-mapping' with a maximum sensitivity loss of ~1.2%. Mapping reads onto a reference genome in this way overcomes an important computational bottleneck in most sequence analysis pipelines (e.g., GATK) [45, 46].

Because CORA identifies redundancies in both sequencing reads and reference data, its computational cost scales linearly with the size of the nonredundant data, which comprise a smaller portion of the total input. Furthermore, CORA constructs a reference homology table data structure (Figures 2-3 and 2-4), which also offers general utility beyond read mapping by providing fast access to all pairs of homologous loci in the reference genome. Moreover, because CORA’s compressive framework achieves speed gains inversely related to the sequencing error rate, the acceleration it provides will substantially improve as sequencers generate higher-quality reads (Figures 2-1, 2-5, and 2-6).
Figure 2-1: Runtime and sensitivity comparison results between conventional read mapping methods and CORA. (a) Runtime comparison results between conventional read mapping methods and CORA for whole-genome gapped and ungapped all-mapping of 1000 Genomes Phase 1 Illumina 2×108 bp paired-end read datasets of one, two, and four Finnish individuals (FIN1, FIN2 and FIN4, with approximately 4×, 8× and 16× read depth-coverage, respectively; graph at left). The mapping similarity threshold is defined as the Levenshtein (edit) distance of 4 for each 108 bp-long read end. For the FIN4 dataset, we additionally performed ungapped mapping experiments with the similarity threshold set as Hamming distance of 4 for each end (graph at right). We compared all-mapping runtimes of Bowtie2 v2.1.0 (with ‘-a’ parameter) and BWA aln v0.7.5a (with ‘-N’ parameter) against compressively accelerated versions of each (CORA-Bowtie2 and CORA-BWA); for the ungapped mapping experiment, we also compared against mrsFAST-Ultra v3.3, which does not perform gapped mapping. We included read dataset compression in the runtime for CORA mappers, but not time to build the homology table; similarly, we did not include genome indexing for other mappers.
To ensure consistency across runtime comparisons, we assumed that all paired-end mappings of a read should be reported individually and consecutively, so that a downstream method can directly use the mapping output. Both CORA mappers and Bowtie2 readily satisfied these criteria; the additional computation needed to ensure this for BWA and mrsFAST-Ultra are indicated with a lighter shade. (b) Sensitivity comparisons indicate that CORA mappers are substantially more sensitive than BWA and Bowtie2 for both gapped (lower) and ungapped (upper) all-mapping. Though it does not have 100% sensitivity like mrsFAST-Ultra, CORA is able to report mapping results with near-perfect sensitivity (~ 99.7%) for ungapped all-mapping. Color key as in a. (c) CORA’s compressive framework achieves speed gains inversely related to the sequencing error rate. The graph shows the runtime of full and coarse ungapped mappings of CORA-BWA when aligning 20 million simulated paired-end reads (100 bp) onto hg19 chromosome 20 at varying sequencing error rates and a fixed mutation rate of 0.1%. ‘Coarse only’ runtime stands for the time required to run BWA within the CORA-BWA pipeline.

In contrast, existing read mappers require a costly sequence-comparison step, called seed-extension, which iteratively maps each read onto a reference genome. The computational cost is high even if the seed-matching step (finding exact matches for short seeds in the reference) is performed simultaneously on identical seeds across the read dataset [47, 48, 49], and the reference is stored, indexed or cached efficiently [40, 41, 44]. Moreover, as whole-read duplicate removal methods [5] require the entire read sequence to be identical, their acceleration is negligible for longer paired-end-read datasets (>75 bp), such as those produced by common sequencing technologies (e.g., San Diego-based Illumina’s HiSeq X and HiSeq 2500). Thus, existing methods’ time requirements scale linearly with the size of the full read dataset, and increase each year along with the exponentially growing read data.

CORA accelerates both gapped and ungapped alignment of conventional best-mapping capabilities of state-of-the-art BWA and Bowtie2 (Figure 2-9). However, CORA’s advance is particularly striking for ‘multi-reads’ (reads that map to multiple locations on the reference genome), enabling massively accelerated all-mapping, even in comparison to state-of-the-art all-mappers (e.g., mrsFAST-Ultra [47]), which achieves improvements in efficiency based on machine architecture, rather than lever-aging redundancy in the data themselves. When CORA-BWA (i.e., CORA framework with BWA plugged-in) computes gapped all-mapping results from a 16× depth-coverage read dataset of four Finnish individuals from the 1000 Genomes Project, it
is >62 times faster than BWA alone, and CORA-Bowtie2 is three orders of magnitude faster than Bowtie2 alone, both CORA versions demonstrating improvements in sensitivity. Our ungapped all-mapping experiments show that CORA-BWA is approximately sixfold faster than mrsFAST-Ultra for human, and approximately ninefold faster for mouse read datasets with minimal loss in sensitivity (<0.4%) (Figure 2-1). Because of CORA’s use of memory-intensive data structures like the homology table, it has relatively higher memory usage (~50 GB) than the other mappers we tested (Tables 2.1, 2.2, and 2.3). However, the added cost of physical memory is unlikely to be a hindrance to large-scale sequencing studies, given the cost-saving benefits, and high sensitivity of accelerated read-mapping using CORA (Tables 2.4, 2.5, and 2.6).

All-mapping is an important component of many downstream analyses. It is the most robust way to comprehensively and accurately analyze structural variants, transposons, copy-number variants and other repeat elements within the genome[50]. There is increasing evidence that using multi-mapped reads enriches the statistical power and accuracy of single-nucleotide polymorphism (SNP) and structural variation discovery [51, 52, 53, 54], transcriptome and isoform quantification [38, 55, 56, 57], and RNA binding-site prediction [58]. Presently, all-mapping, in particular gapped all-mapping, is often not used by sequence analysis pipelines because of its high computational cost. CORA’s ability to compressively accelerate existing mappers to achieve sublinear time-scaling enables use of all-mapping in large-scale sequence analysis pipelines (Figure 2-8).

As state-of-the-art NGS technologies continue to improve and generate evermore quantities of data with higher quality, the amount of redundant sequence information within them also increases. Compressive methods that can leverage this type of redundancy will have an important part to play in how the biomedical community tackles the sequencing data deluge in the upcoming years.

2.2 Methods

CORA’s acceleration relies on three key components (Figure 2-2):
1. Identifying redundant k-mers across reads in the input read dataset first, as opposed to mapping each read directly to the reference.

2. Mapping the reference to itself through a single preprocessing step, in order to create a comprehensive lookup table of similarities for fast retrieval.

3. Using a local neighborhood search in the Hamming and Levenshtein distance space of the self-mapped reference to speed up sequence comparison.

The CORA framework is comprised of combining these advances with an off-the-shelf mapper for identifying inexact k-mer matches in the reference.

For preprocessing, we map the reference onto itself by building a high-resolution homology table of the reference genome sequence. The table stores the similarity information of all homologous or similar regions of a specified length in the reference genome, within a predefined Hamming distance. With this table, we can directly access all homologous or similar loci of any given locus, and thus are able to report all possible mapping locations at low computational cost, which is particularly useful for multi-reads (reads that map to multiple locations in the reference genome).

For the actual read mapping stages, we first map the reads to themselves in order to identify and compress shared k-mers, with the goal of constructing a compressed representation of the read dataset consisting of only unique k-mers. We then use an off-the-shelf short-read mapper (e.g., BWA [40] or Bowtie2 [41]) to coarsely map this compressed k-mer dataset to k-mer matches in the reference genome; we represent these matches in the form of links, or pointers, to the reference genome. Lastly, we generate the final set of mappings through simultaneous traversal of the homology table and multiple k-mer links of the read. We are thus able to identify all high-quality alignment positions in the reference with near-perfect sensitivity.

In the following subsections, we present the technical details of these advances.

2.2.1 Homology table of the reference

Before performing compressive read mapping, we first preprocess the reference to generate the homology table, a high-resolution all-to-all similarity map of the ref-
Figure 2-2: Conventional read mapping methods and compressively accelerated read mapping framework. (a) In order to generate read-mapping results, existing read mappers compare each read to the reference or a previously-constructed index for the reference. (i) Unique-read case: Some reads have a unique match to the reference. (ii) Multi-read case: Due to the repetitive structure of DNA sequences, a single read can often be aligned to more than one location with high similarity in the reference. Most existing approaches involve costly seed-extension or suffix-array traversal stages for each of these locations, requiring additional computational time reporting multiple mappings for each read (or comparing them to report a best-mapping). Furthermore, in high depth-coverage datasets (especially when multiple individuals are mapped together), there can often be reads that are fully or partially similar to other reads in the dataset. (iii) Redundancy within reads: As existing aligners cannot utilize inexact redundancy within reads, they process each read individually, potentially duplicating previous computations performed for earlier reads in the dataset. This inefficiency is particularly an issue for multi-reads sequenced from highly-repetitive regions in the genome, since they require sequence comparisons with a large number of loci in the reference. Compressively accelerated read mapping addresses the inefficiencies of mapping high-throughput NGS reads by capitalizing on redundancy within both read datasets and the reference.
Figure 2-2: (cont.) (b) CORA capitalizes on redundancy in both reads and reference. (1) As a preprocessing step, a high-resolution homology table is created for the reference sequence by mapping the reference to itself. The homology table contains all homologous pairs of loci in the reference above a similarity threshold, allowing fast direct access to similar locations in the reference during mapping. (2) The first step in compressive read-mapping is to compress the reads in order to eliminate full or partial redundancies across reads in the dataset. Compression is achieved through self-mapping of the read dataset. (3) Next, an off-the-shelf aligner can be used to perform a coarse mapping from the compressed read data (clusters of similar substrings) to the reference. (4) Each read link represents a cluster of substrings from one or more reads in the dataset and stores their differences from a locus in the reference. (5) Read links are further expanded to obtain final mapping results through traversal of the precomputed homology table, and final mapping results are reported. (6) Far fewer comparisons are required for compressive read mapping due to efficient utilization of redundancy within read sequences as well as the reference.

For a chosen homology block length (e.g., 33-64 base pairs) and similarity measures (e.g., Hamming distances 0-3), the homology table of a reference genome contains links between all homologous loci satisfying the length and similarity requirements. In other words, instead of representing the sequences of k-mers in a given reference genome, a homology table represents pointers for each position in the genome indicating all of its neighboring k-mers. As such, the homology links form an inter-web of similar loci within the reference genome that enables CORA’s k-mer-based compressively accelerated read mapping framework. Though relatively costly to generate (~18 hours for constructing a k=54bp homology table of the hg19 human reference genome with a Hamming distance of 2 on a 12-CPU Intel Xeon X5690 machine using 24 parallel threads), both homology table and reference index need be computed only once for a given reference genome, after which they can be repeatedly used for further compressive mapping runs on the same reference genome.

More formally, given a reference DNA sequence $R$, a substring of length $k$, and a
mismatch error threshold $s$, a homology table $H(R, k, s)$ is a compact data structure that stores the links among all similar $k$-base pair substrings ($k$-mers) of $R$ (up to $s$ mismatches), including reverse complements. To efficiently construct the homology table $H$, we first build an exact homology table $HE(R, k)$ that stores the links among all identical $k$-mer occurrences in the reference under reverse complementation. Based on this exact homology table, we construct an inexact homology table $HI(R, k, s)$ that stores the homology links among $k$-mers with at least one and at most $s$ mismatches (Figure 2-3). Together, the exact homology table $HE(R, k)$ and the inexact homology table $HI(R, k, s)$ form $H(R, k, s)$. In other words, instead of storing neighborhood information for all positions in the genome, CORA compactly represents the neighborhood information between sets of identical $k$-mers.

The exact and inexact homology tables are generated, organized and stored separately in order to optimize runtime performance and memory/disk usage. Though we describe below a substitution-only version of the inexact homology table, it is straightforward to extend homology tables to other types of edits such as indels.

**Exact homology table**

We use an unordered hash table to construct the exact homology table. For each $k$-mer, we use its sequence as the key and its first occurring position in the reference genome as the value in the hash table. If the $k$-mer appears in other position(s) in the reference, an equivalence class is created to store all such positions. Each equivalence class is stored as a list of genomic positions and their directions with respect to the representative, which allows a quadratic number of exact homology links between pairs of genomic loci to be stored within linear space. This representation is further compacted by collapsing adjacent concordant equivalence classes. If there are two or more adjacent concordant equivalence classes (all loci within an equivalence class are shifted by one position with respect to another equivalence class), they are merged in a way that only the values of the first equivalence class with the smallest representative position are kept, with an additional block-length value, which indicates how many adjacent concordant equivalence classes are collapsed into a single one.
Figure 2-3: Compact representation of homology table with exact and inexact homologies. Identical homology blocks of a certain length are collected under the same equivalence class. Each equivalence class has a representative locus, as well as other loci that are oriented with respect to the representative locus. Inexact homology table is a graph with equivalence classes as nodes and partial similarities between equivalence classes as edges. Therefore, inexact homologies can only be defined between equivalence class representative loci. Each inexact homology edge identifies the offset from the beginning of the first equivalence class, the offset from the beginning of the second equivalence class, the length of the inexact homology block, forward or reverse-complement direction of homology, and the positions of differences or base substitutions (text above bold bi-directional arrow). An inexact homology block size indicates the final length of the inexact homology after two or more consecutive and concordant inexact k-mer homologies are merged together. Two consecutive inexact k-mers are defined as concordant if their target positions in the reference are also consecutive and their edit positions in the k-mer are concordant, i.e., containing the same (k - 1)-mers.
A formal definition of the exact homology table block merging can be given as follows: Denote $e_i = (c_i, p_i, d_i)$ as members of the equivalence class $E$, for $i$ ranging from 0 to $|E| - 1$, $c_i$ representing the chromosome ID of the $i^{th}$ element, $p_i$ being a positive integer representing the chromosome position of the $i^{th}$ element, and $d_i$ being either 1 or $-1$, respectively representing the forward or reverse complement direction of the homology of the $i^{th}$ element, with respect to the representative element $e_0$. The representative element is required to always have forward direction ($d_0 = 1$).

Two equivalence classes $E$ and $F$ are merged to create a new equivalence class $E_2$ of block length 2 if and only if $|E| = |F|$ and for all $e_i = (c_i, p_i, d_i) \in E$, there exists an $f_j = (c_j, p_j, d_j) \in F$, such that $c_i = c_j$, $d_i = d_j$, and $p_j = p_i + (d_i \times d_j)$. Blocks of length longer than 2 are inductively defined: If $E$ and $F$ can be merged, $F$ and $G$ can be merged, then $E$, $F$, and $G$ can be merged altogether into $E_3$, an equivalence class with a block length of 3, and so on.

**Inexact homology table**

Generating the inexact homology table, $HI(R, k, s)$, which represents all-to-all homologies of $k$-length substrings in $R$ with between 1 and $s$ mismatch errors involves a similar but more sophisticated approach. In order to generate the inexact homology table $HI(R, k, s)$, we use the exact homology table $HE(R, k)$, specifically the precomputed equivalence classes, to reduce computation and storage requirements. Instead of constructing the inexact homologies between substrings from the reference, we need only do so among the equivalence class representatives in the exact homology table $HE(R, k)$. The remainder can be inferred directly using the equivalence class members. The inexact matches between equivalence class representatives are identified by a seed-and-extend procedure, which first checks if there is an exact match between seeds on two given $k$-mers, and then extends the remaining bases of the $k$-mer sequences to identify any mismatches. We use a special spaced-seeding scheme for performance and parallelization purposes (see 2.2.1 Seed position selection scheme).

To construct $HI(R, k, s)$ for a given $R$, $k$ and $s$, we construct $s + 1$ auxiliary hash tables, each corresponding to one of $s + 1$ seeds extracted from equivalence classes.
from the exact homology table. The seeds are sampled according to the position selection scheme described above. In the hash table, we use the binary representation of a seed as the key, and the list of positions in the reference that contain the seed sequence as the value. Every time we find a hit, the current k-mer position in the reference is added to the list of the corresponding key.

Finally, we further compact the representation of the inexact homology table by merging any adjacent pairs of inexact matches with consecutive mismatch positions in forward and reverse complement alignment directions, similar to what we did for the exact homology table, but with the added requirement of concordant mismatch positions that are shifted by one.

Example homology table construction

In order to elucidate the structure of the homology tables, we provide below a toy genome example with a single chromosome and construct a homology table with a k-mer length of 4bp and a Hamming distance threshold of 1.

\[ \text{ACCGCAGCGGT} \]

We initially build an exact homology table (HE) from the genome, identifying each 4-mer or its reverse complement, if lexicographically earlier. These correspond to:

\[ 1^+ : ACCG, \quad 2^+ : CCGC, \quad 3^+ : CGCA, \quad 4^- : CTGC \]
\[ 5^+ : CAGC, \quad 6^+ : AGCG, \quad 7^- : CCGC, \quad 8^- : ACCG \]

with each number representing the position of the 4-mer in the genome and the \((+)/(−)\) sign indicating whether the default or the reverse complement 4-mer is considered. Note that positions 4, 7 and 8 have a \((-)\) sign after them since their reverse complement 4-mers are lexicographically earlier than the forward ones.

Once the 4-mers are identified, identical 4-mers are merged into groups in order
to construct the equivalence classes. This grouping results in 1+ and 8− in one
equivalence class whereas 2+ and 7− are in another; all the remaining positions
without identical 4-mers each have their own equivalence classes, i.e., which contain
only themselves. Each equivalence class has a single representative, which is always
in the forward direction (otherwise, all elements are in reverse complement to ensure
this property), and other elements in the equivalence class assume directions relative
to the representative. In our toy genome example, all six equivalence classes are:

\([1 : 8−], \ [2 : 7−], \ [3], \ [4], \ [5], \ [6]\)

where the first number indicates the representative position, and the remaining
numbers, non-representative equivalence class members with a direction in relation
to their corresponding representatives. In order to conserve space, single item classes
are not stored in the exact homology table \(HE\), leaving \([1 : 8−]\) and \([2 : 7−]\).

To compact the exact homology table, all adjacent concordant equivalence classes
are merged into block homologies that represent intervals longer than 4-mers. In our
toy genome example \([1 : 8−]\) and \([2 : 7−]\) are adjacent (since 1 and 2 are adjacent)
and concordant (since other elements in the two equivalence classes, 8− and 7− are
also adjacent in reverse direction). We represent the merged exact homology block in
the genome with an additional block length parameter appended to the equivalence
class, in this case \([1 : 8−]\)(2). The set of all compact equivalence classes of size larger
than one (with a block length 1 or longer) constitutes the exact homology table, \(HE\).

Note that the block merge operation can also be done with forward direction ele-
ments as well as with multiple items, as long as adjacency and concordance properties
are satisfied. For example, an equivalence class \([15 : 35+, 73−]\) can be merged with
\([16 : 36+, 72−]\) to form block equivalence class \([15 : 35+, 73−]\)(2), whereas not with
\([16 : 36+, 74−]\). Similarly, multiple equivalence classes can be merged together. For
example, \([10 : 20+], \ [11 : 21+], \ [12 : 22+]\) and \([13 : 23+]\) can be merged into a single
block equivalence class \([10 : 20+]\)(4).

In order to construct the inexact homology table, \(HI\), we input the set of 4-mer
equivalence classes formed during HE construction before we perform any block merge operations or elimination of single-item equivalence classes, which is:


Among these, we identify all pairs of inexact homologies between class representatives within a Hamming distance of 1. These six homologies are:

\[ [1 \rightarrow 6+ : 2] \quad [2 \rightarrow 5+ : 2] \quad [3 \rightarrow 6- : 4] \]
\[ [4 \rightarrow 5- : 3] \quad [5 \rightarrow 7- : 2] \quad [6 \rightarrow 8- : 2] \]

where the first number represents the source position; the second number with the appended sign, the target and direction of homology; and the number after the semicolon, the 1-base error positions at which the two 4-mers differ, oriented by the direction of the first 4-mer. The reverse homologies ([6 \rightarrow 1+ : 2], [8 \rightarrow 6- : 3], etc.) are not represented in HE, as they are redundant.

Similar to merging adjacent equivalence classes, we can also merge adjacent and concordant inexact homologies. In the case of inexact homologies, the concordance property requires the error positions to be concordant as well. For example, the homologies [3 \rightarrow 6- : 4] and [4 \rightarrow 5- : 3] are adjacent and concordant, since 3 and 4 as well as 6- and 5- are adjacent, and the error positions 4 and 3 are concordant with the adjacent 4-mers. In order to compact the inexact homology table we can represent this inexact block homology as [3 \rightarrow 6- : 4](2), where 2 refers to the number of homologies merged. This example represents an inexact homology between the 5-mer that starts at position 3 (CGCAG) and the 5-mer, at position 5 (CAGCG), in reverse complement direction with a mismatch error at nucleotide 4. Similar to compacting HE, more than two HI homologies can be compacted as well as inexact homologies with multiple errors, as long as each error position satisfies the concordance property.

Thus, the set of homologies in HI for our toy genome example will be:
Figure 2-4: Seed position selection for inexact homology table construction. This seed selection scheme allows detection of all inexact homologies of a given k-mer length within a Hamming distance of 2 (by pigeonhole principle). For detection of higher numbers of mismatches, the number of seeds sampled can be increased. Rather than selecting consecutive positions for each seed, spacing out seed positions throughout the k-mer allows for more evenly-sized bins within the hash table. The positions sampled for each seed is designed to be closed under reverse complementation, so that hash tables created for each seed are disjoint. This allows for both space savings and easy parallelization of inexact homology table construction.

HE and HI together constitute the homology table of the toy genome.

Seed position selection scheme

For inexact homology table construction, we use a special spaced-seeding scheme closed under reverse complementation for performance and parallelization purposes, as displayed in (Figure 2-4).

When determining seeds for inexact homology table construction, we choose seed positions to be closed under reversal for two particular reasons. First, it allows the reverse complement of a seed to correspond to the exact same positions in the k-mer; thus we need only store the lexicographically smaller direction of any seed in the hash table and search for each seed once in the hash table. Second, this choice allows both simple parallelization and multi-pass implementation of the hash table, since data splitting can be performed easily without the need to search for a seed and its reverse complement in different hash tables. This choice allows for both space savings and ease of parallelization for inexact homology table construction.

\[ [1 \to 6+: 2], [2 \to 5+: 2], [3 \to 6-: 4](2), [5 \to 7-: 2], [6 \to 8-: 2] \]

HE and HI together constitute the homology table of the toy genome.
Furthermore, as DNA sequences are highly repetitive, some bins in a hash table can get very large for frequently-occurring seeds. Rather than selecting consecutive positions for each seed, spacing out seed positions throughout the k-mer allows for more evenly-sized bins within the hash table. For this reason, we choose a spaced seeding scheme in which each position within a seed is sampled as far away as possible while maintaining that the positions are closed under reversal (Figure 2-4).

We experimentally evaluated the fitness of our seed position selection scheme by comparing it to a variety of schemes that are also closed under reversal. The schemes were based on selecting:

1. Equally-spaced nucleotides from the k-mer which is built in CORA.
2. Equally-spaced dinucleotides of a seed rather than each nucleotide
   (e.g. 112233112233 instead of 123123123123)
3. Equally-spaced 4-mers (e.g. 111122223333)
4. Equally-spaced 8-mers
5. Random selection of seed positions that are not equally-spaced, satisfying only that seeds have equal length.

We created the inexact homology table for chr20 of hg19 using each scheme three times. We noted that the slowest scheme was (4) with an average runtime of 680 seconds. As equally-spaced seed sub-blocks got shorter, the runtime also improved resulting in 639 seconds for (3), 568 seconds for (2), and 466 seconds for (1), which is the scheme we eventually used for CORA’s homology table construction algorithm, which equally spaces all nucleotides of each seed. The randomly generated schemes in (5) were faster than (2) but slower than (1), averaging 557.6 seconds, with the best performing random scheme taking 549 seconds.

**Homology table implementation and memory use**

For smaller genomes, CORA’s homology table construction algorithm can be run on the entire genome, processing all k-mers of the genome in a single pass. However, for
larger genomes (e.g., mouse and human), the memory requirements for construction is
expensive. Once the homology links are compacted, a homology table for the human
genome can be reduced to a manageable size of ~15 GB; however the memory usage
for the intermediate data structures would be on the order of several hundred GBs.
In order to enable homology table construction for smaller memory machines, CORA
employs a multi-pass homology table construction algorithm.

The basic idea behind the multi-pass construction of the homology table is to
separate the total $k$-mer space into disjoint sets that are closed under reverse com-
plementation, so that each pass of the multi-pass construction scheme can detect the
exact homologies within the current set. For the sake of simplicity, we separated the
set of $k$-mers based on the nucleotides on the two ends of the $k$-mer sequence. Note
that if one end is used as such a separator for subsets, the other end also needs to be
used due to reverse complementation. For instance, $k$-mers that start with $A$ should
be in the same subset as $k$-mers that end with ‘T’, requiring at least two nucleotides
to be used for the distribution of the $k$-mer. For instance, $k$-mers that start with
$A$ and end with $G$ (in the form $[A...G]$) should be processed in the same pass with
the reverse complement form $[C...T]$. For these $k$-mers, we construct the $A-G$ hash
table. In this way, we can construct the whole table with 10 passes. Specifically, these
10 passes would include: 1) $A - A$ and $T - T$; 2) $A - C$ and $G - T$; 3) $A - G$ and
$C - T$; 4) $A - T$; 5) $C - A$ and $T - G$; 6) $C - C$ and $G - G$; 7) $C - G$; 8) $G - A$
and $T - C$; 9) $G - C$; and 10) $T - A$. By incorporating more nucleotides from both ends
of the $k$-mers, it is possible to separate the full $k$-mer space into smaller subsets and
thus store the hash-table using even smaller memory size during each pass.

For the mapping of FIN datasets presented, we constructed the inexact homology
table for the hg19 reference genome with a 10-pass approach with interleaved 24-split
parallelization on 12 hyper-threaded processors. For parallelization of split signals,
two bases from the ends of each seed are used, which gives 136 reverse complement
agnostic sets, as opposed to 10 for one base from each end. These are collected into 24
balanced-set groups. The balancing is accomplished according to the general human
GC ratio assumption $2 \times |A \text{ or } T| = 3 \times |C \text{ or } G|$. 

47
Table 2.1: Runtime and memory cost of CORA’s homology table construction stage for different multi-pass construction schemes.

<table>
<thead>
<tr>
<th>Number of passes</th>
<th>Peak memory use (GB)</th>
<th>Elapsed time (12-CPU hours)</th>
<th>Total CPU hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>94.5</td>
<td>18.2</td>
<td>181.8</td>
</tr>
<tr>
<td>8</td>
<td>87.2</td>
<td>18.5</td>
<td>180.1</td>
</tr>
<tr>
<td>10</td>
<td>81.3</td>
<td>18.8</td>
<td>170.1</td>
</tr>
<tr>
<td>24</td>
<td>50.1</td>
<td>21.3</td>
<td>165.5</td>
</tr>
<tr>
<td>48</td>
<td>44.4</td>
<td>25.7</td>
<td>171.5</td>
</tr>
<tr>
<td>136</td>
<td>40.5</td>
<td>30.6</td>
<td>192.7</td>
</tr>
</tbody>
</table>

When generating exact and inexact homology tables, CORA allows the user to specify the number of passes in the multi-pass construction scheme as well as the number of processes to be used for parallelizing each pass.

Below is a runtime/memory tradeoff analysis for construction of the homology table for the hg19 human reference genome for 54-mers with a Hamming distance of 2. This is the homology table we used for the FIN1-FIN4 read mapping experiments presented in Figure 2-1. We carried out 5 different levels multi-pass runs comprising 6, 8, 10, 24 or 48 passes. The 6, 8 and 10-pass runs were performed using a 2-nucleotide signal (1 from each end of the k-mer), whereas the 24 and 48-pass runs were performed using a 4-nucleotide signal (2 from each end of the k-mer). For each run we present the peak memory used by the operating system (maximum memory resident set size), total process runtime (user + system), as well as the elapsed time using 12 hyper-threaded CPUs (Table 2.1).

We observe that peak memory usage decreases with more passes for constructing the homology table, since the computation is done in smaller chunks. The total process time also decreases with the number of passes due to smaller hash-tables which are less costly to maintain. On the other hand, the elapsed time increases as the time spent for I/O is higher for multi-pass runs due to the requirement that both the reference genome and the intermediate files need to be fully scanned for each pass, and these scans are not parallelized.
2.2.2 Query-side compression and coarse mapping

In this phase of the CORA framework, the main goal is to reduce the redundant information within and across multiple large NGS read datasets, achieving a compact representation of the reads in the form of k-mer links to the reference; these represent a position in the reference genome similar to a substring within the read and the differences of the read from it. This representation of reads allows faster sequence matching as we need to check only corresponding edits during the homology table traversal stage rather than performing expensive sequence comparisons. The two main stages of the links table construction are (1) collapsing and (2) coarse mapping.

K-mer collapsing

The aim of the collapsing stage is to eliminate k-mer redundancies in the read dataset; this stage inputs a set of FASTA/FASTQ read datasets and converts them into a set of non-redundant k-mers.

In a simple scenario in which the reads are single-end and read length is very short (<50bp), we can assume the k-mer length to be equal to the read length. In this simple scenario, the k-mer collapsing scheme would simply involve identification of all identical reads (under reverse complementation) and collapse them into unique reads with compact IDs which encode the set of reads they correspond to in the input FASTA/FASTQ file. These IDs comprise a range of text-readable ASCII characters from ‘!’ to ‘~’ (with an alphabet size of 94), allowing off-the-shelf mappers to readily parse the read name format without any modifications and directly operate on CORA’s compact read representation.

A direct whole-read collapsing approach as described above can be beneficial in terms of mapping speed for very short single-end read datasets (<60 bp); however, as the chances of a read to not contain any sequencing errors reduces drastically the longer the reads get (~22% for 75bp and ~5% for 150bp with a 2% sequencing error rate), the potential speed gain from whole-read compression also reduces drastically since even a single sequencing error in the read would virtually eliminate the chances
of the read being an exact duplicate. The speed gains reduce even further when the reads are paired-end, as whole-read compression would require both mates to be merged together as a single $k$-mer for collapsing, resulting in even lower probability of the read being an exact duplicate.

In order to improve read $k$-mer collapsing performance of longer NGS reads (>70bp), as well as paired-end reads, instead of compressing the entire read sequence, the CORA framework splits the (paired-end) read into shorter $k$-mers of the same length (between 33-60 base pairs) and collapses them as independent sequences, producing a set of non-redundant $k$-mers. This collapsing stage is applied prior to the coarse mapping stage, and therefore the subsequent coarse mapping step is performed in a read agnostic manner (i.e., without the knowledge of which $k$-mer sequences come from which reads). For the $k$-mers of a read (and mates of a paired-end read) to be identified after coarse mapping, CORA assigns corresponding IDs to the splits and mates of the same read. Only after the coarse mapping stage are these collapsed $k$-mers reorganized based on their IDs and merged into whole reads in order to produce the final mapping results described in 2.2.3 Homology table traversal. As a tool that can rapidly identify the positions of homologous $k$-mers in the reference, the homology table precisely complements the $k$-mer based compression of reads, together creating a very efficient method for NGS read mapping for large datasets with high redundancy.

The length of each sequence ID in the collapsed $k$-mer dataset is determined by the total number of reads to be mapped in a single mapping task. The compact encoding scheme CORA uses allows each read in a dataset consisting of ~19.5 million paired-end reads to be uniquely identified with 4 character sequence IDs, ~1.8 billion paired-end reads with 5 characters, and ~172 billion paired-end reads with 6 characters. As the sequence IDs are assigned the same order as they are listed in the input FASTA/FASTQ files, CORA also can optionally retrieve the original read name, quality scores, and sample information.

In addition, through the use of a database of $k$-mers in the reference, CORA is able to identify exact $k$-mer matches to the reference during collapsing and print
them to a different output together with the chromosome and position information of the reference $k$-mer match, further reducing the number of non-redundant $k$-mers to be processed in the coarse mapping stage. This optional feature is enabled in the experimental results presented.

Apart from compressive-mapping speed gained from $k$-mer based read compression, the flexibility of collapsing independently chosen $k$-mers within the read rather than the whole read enables a number of advantageous features of CORA’s framework. As the $k$-mer length is independent from the read-length, reads with different lengths can be processed together in the same compressive mapping run. Independently processing $k$-mers also allows dynamic trimming of low quality reads on-the-fly during mapping, which further facilitates compression by reducing parts of the reads with lower redundancy. One caveat is that when overlapping $k$-mers are selected, the maximum mapping distance can be reduced if there are variants or sequencing errors in the overlapped region in the read. One solution to this potential problem would be to always pick non-overlapping $k$-mers in the reads, but leave a small number of bases in the read that are not covered by $k$-mers; these bases then can be checked during the mapping inference stage in order to verify whether the mapping similarity requirements are satisfied when considering the uncovered bases.

Selecting fewer but longer $k$-mers results in lower compression in the collapsing stage, increasing the time spent on coarse mapping; but longer $k$-mers decrease the average number of neighbors each reference locus has in the homology table, enabling faster homology table traversal. On the other hand, selecting many but shorter $k$-mers results in higher compression during collapsing and less time spent on coarse mapping; but shorter $k$-mers increase the number of homologies in the homology table, as well as time spent on homology table traversal.

Coarse mapping

Coarse mapping is the stage in which the CORA framework utilizes the short-read mapping capabilities of an off-the-shelf mapper. Any read mapping tool that can report a best-mapping (i.e., a mapping with a minimum Hamming or Levenshtein
(edit) distance among all possible mappings) of a read, such as BWA and Bowtie2, can be directly adapted to the CORA framework. Theoretically, CORA does not require a best-mapping to be found. Finding any mapping within the distance threshold would suffice to guarantee perfect sensitivity assuming the homology table distance threshold is twice the search distance threshold. However, when the homology table is imperfect (the distance of the homology table is less than twice the distance threshold used for mapping), CORA benefits from coarse mapping being performed by a best-mapper.

For off-the-shelf aligners that do not allow a best-mapping scheme, alternative strategies can be designed for integration with some caveats. If a mapper does not allow best-mapping but allows a user-specified limit on the number of reported mappings, a mapping mode that reports only a single mapping (not necessarily with minimum error) can be integrated into CORA, with minor loss in final read-mapping accuracy. If the off-the-shelf mapper provides user-specified limits on the number of errors within a read, it is also possible to perform an iterative re-alignment scheme, at each stage aligning previously unmapped reads with a higher error threshold. While this approach will preserve final read-mapping accuracy, it will yield less compressive acceleration due to the heavy cost of re-alignments within the coarse mapping stage.

Unlike most read mappers, CORA's coarse mapping stage does not aim to find actual mapping locations for each read's k-mers (which would ultimately provide little compressive acceleration), but instead identifies only a single good 'representative' position in the reference that a k-mer can be mapped to and be represented in the form of compact links (i.e. position and differences). For short k-mer sequences that are used within the coarse mapping stage (33-64bp), it is substantially faster to find a single good location in the reference than find all mappings within the substitution-error neighborhood.

For paired-end mapping, finding a proper mapping in the reference involves finding multiple good locations for each mate and merging them according to the user-defined allowable interval of insert size between the mates (e.g. 150-650bp). Coarse mapping for a paired-end read consists of merely finding a good location in the reference for
each mate independently. Similarly, for reads that are divided into multiple k-mers, each k-mer is coarse mapped independently to the reference genome.

Through these conceptual advances in the mapping stage, the CORA framework achieves massive speed gains (typically 1-3 orders of magnitude for all-mapping and more than three times the speed-up for best-mapping), as compared to the original performance of the off-the-shelf mappers.

Constructing the links table from the coarse mapping output generated by the off-the-shelf mapper (in SAM format) involves simply scanning the SAM file and creating a link item for the read code on each line. Each link in the table contains the read name encoding described above, the position in the reference, as well as a list of mismatch offset positions from the beginning of the read together with its nucleotide change in the read. Since the links table contains the differences of each k-mer from the reference, it is possible to regenerate the original read sequences in the input, and thus compression of read data within the CORA framework is not lossy.

**Query-side compression and complexity of coarse-mapping**

We described above that query-side compression consists of grouping shared k-mers across the multi-individual read dataset, which then are coarse mapped onto the reference genome. In order to estimate the speed-up directly gained from k-mer compression of reads, we analyze the number of unique k-mers present in a large set of reads drawn from multiple individuals’ genomes. For simplicity of analysis, we will initially assume the read dataset contains a single haploid individual and the sequencer returns k-mers rather than full reads. We will later generalize the model to include multiple potentially overlapping k-mers in each read from multiple diploid/polyploid individuals.

*Single k-mer read model.* Let $\Sigma = \{A, C, G, T\} \cong \mathbb{Z}/4\mathbb{Z}$ be the alphabet from which the k-mer bases are drawn. Define $0.9 < p < 1$ and $q = 1 - p$, such that a base is read correctly with probability $p$ and incorrectly with probability $q$. For simplicity, we assume that a sequencing error is equally likely among the three alternative nucleotides.
Let the random variable \( \sigma : [0, 1] \to \Sigma \) be defined by

\[
\sigma(\omega) = \begin{cases} 
0, & \text{if } \omega \in [0, p) \\
1, & \text{if } \omega \in [p, p + q/3) \\
2, & \text{if } \omega \in [p + q/3, 1 - q/3) \\
3, & \text{if } \omega \in [1 - q/3, 1]
\end{cases}
\]  

(2.1)

Therefore, \( \forall \ell \in \Sigma, l + \sigma = l \) with probability \( p \). This is to say that a base is read correctly with probability \( p \) and incorrectly with probability \( q \).

Given \( x \in \Sigma^k \), define \( x_i \) as the \( i \)th letter (base) of \( x \). For all \( x \in \Sigma^k \), define independently the \( \Sigma^k \)-valued random variables \( R^x \) by \( \forall i, R^x_i = x_i + \sigma_i \), where \( \sigma_1, \ldots, \sigma_k \) are independent instances of \( \sigma \). Thus, \( R^x \) can be thought of as a read of \( x \), including machine errors.

We will model the sequencer by a list \( R^{y_1}, \ldots, R^{y_c} \), where \( c \) is the total number of \( k \)-mers read and each \( y_i \) is drawn uniformly randomly from the set \( G' \) of unique \( k \)-mers in \( G \). We will make the simplifying assumption that elements in \( G' \) are uniformly randomly distributed in \( \Sigma^k \). We will eliminate both of these assumptions later on.

Let \( K \) be the set of unique \( k \)-mers in the list \( \{ R^{y_1}, \ldots, R^{y_c} \} \). For some particular \( y \), if \( \delta(x, y) = d \), the Hamming distance of \( x \) from \( y \), then \( \mathbb{P}(R^y = x) = p^{k-d}q^d3^{-d} \) because each of the matched locations must be preserved, each of the mismatched locations must be altered, and the chance of a correct alteration at each mismatched location is \( 1/3 \). Thus, if \( y \) is randomly sampled from \( G' \), by the union bound (Boole’s inequality), we can conclude

\[
\mathbb{P}(R^y = x) \leq \sum_{d=0}^{k} p^{k-d}q^d3^{-d} \cdot \mathbb{P}(\delta(y, x) = d).
\]

By our assumption that non-redundant \( k \)-mers in the genome \( G \) are distributed uniformly and continuously in \( k \)-mer space,

\[
\mathbb{P}(\delta(y, x) = d) \approx \frac{\binom{k}{d} \cdot 3^d}{4^k}.
\]
Consider now the total contribution from \( k \)-mers in the genome that are distance \( d \) from \( x \) versus the total contribution from those that are distance \( d + 1 \) from \( x \):

\[
\frac{p^{k-d}q^d \cdot \binom{k}{d}}{p^{k-d-1}q^{d+1} \cdot \binom{k}{d+1}} = pq^{-1} \cdot \frac{d}{k-d}.
\]

This implies that the contribution from \( k \)-mers further away from \( x \) decreases exponentially with distance, so long as \( q < 1/k \). Thus, \( \mathbb{P}(R^y = x) \approx p^{k-d}q^d \cdot \binom{k}{d} \cdot 4^{-k} \), where \( d = \delta(x, G) \). This implies that \( \mathbb{P}(x \in K) \) is largely determined by \( \delta(x, G) \) and by the total number \( c \) of \( k \)-mers read:

\[
1_x \approx 1 - \left(1 - p^{k-d}q^d \cdot \binom{k}{d} \cdot 4^{-k}\right)^c,
\]

where \( d = \delta(x, G) \) and \( 1_x \) is an indicator random variable for \( x \in K \).

By linearity of expectation,

\[
E[|K|] = E \sum_{x \in \Sigma^k} 1_x \approx \sum_d \left[ 1 - \left(1 - p^{k-d}q^d \cdot \binom{k}{d} \cdot 4^{-k}\right)^c \right] \cdot |x \in \Sigma^k : \delta(x, G) = d|.
\]

We can bound \( |x \in \Sigma^k : \delta(x, G) = d| \leq \binom{k}{d} \cdot 3^d \cdot |G| \). Additionally, if \( G \) is uniformly distributed across \( \Sigma^k \), this bound is also a good approximation when \( \binom{k}{d} \cdot 3^d \cdot |G| < 4^k \) because the Hamming shells of radius \( d \) around each point in \( G \) intersect minimally. Thus, we can estimate the number of unique \( k \)-mers

\[
E[|K|] \approx \sum_{d=0}^{D} \left[ 1 - \left(1 - p^{k-d}q^d \cdot \binom{k}{d} \cdot 4^{-k}\right)^c \right] \cdot S_d,
\]

where

\[
S_d = \binom{k}{d} \cdot 3^d \cdot |G|
\]

for \( 0 \leq d \leq D - 1 \), and

\[
S_D = 4^k - (S_1 + \cdots + S_{D-1}).
\]

This bound indicates that Hamming shells with shorter radii around \( k \)-mers sam-
pled from the genome are saturated exponentially faster than larger radii, causing the set of unique $k$-mers to grow sublinearly even in the presence of sequencing errors.

To find the coarse mapping speedup from query-side compression in the single $k$-mer read scenario, we simply take $c/E[E[|K|]]$, which is equivalent to the rate of redundancy. In the case that the non-redundant data scales sublinearly with the full data, as it is for most real-life biological datasets, the coarse mapping time of CORA will also be sublinear in the number of reads.

Note that the assumptions that $G'$ consists of only unique $k$-mers and that $k$-mers in $G'$ are uniformly randomly distributed both correspond to the worst case scenario for the upper-bound on the number of unique $k$-mers:

1. If the $k$-mers were drawn from a multi-set of non-unique $k$-mers in the genome, sequenced $k$-mers would have a higher likelihood to be redundant on average compared to a set that contains the same $k$-mers with no duplicates.

2. If $G'$ is not uniformly randomly distributed (e.g. there are inexact repeats in the genome), the shorter-radius Hamming shells of some $k$-mers in $G'$ will overlap with an increased likelihood. Therefore the chances of each sampled $k$-mer with sequencing errors to be a unique $k$-mer is reduced.

**Multiple $k$-mers per read.** We can generalize the single $k$-mer read model described above to multiple $k$-mers per read, since contiguous $k$-mers in the read are also contiguously sampled from the genome. In the case that two $k$-mers overlap in the read, any sequencing error in the overlap region will be observed jointly in two separate $k$-mers. As this reduces the combination of possible $k$-mers that can be drawn from these two $k$-mers with a joint sequencing error distribution, the expected rate of unique $k$-mers will also be lower compared to the scenario in which the two $k$-mers are disjoint with independent sequencing error distributions.

Generalizing the model above to diploid/polyploid genomes from multiple individuals with SNPs, indels and other variants requires altering the unique $k$-mer set $G'$ to encompass all of the unique $k$-mers in the complete genomes of the individuals sequenced. In a real biological population of genomes of the same species, it is ex-
Figure 2-5: Estimation of redundancy within read datasets in absence of sequencing error. The plot above demonstrates how the number of k-mers processed by the coarse mapping stage scales with respect to total number of reads in the input dataset, for a high depth-coverage simulation of 100bp paired-end reads on hg19 chromosome 20, with 0.1% mutation rate and 0% sequencing error. After roughly 100 million reads in the input dataset, additional reads in the dataset do not affect the size of the coarse mapping stage, thus resulting in sublinear mapping scalability with the CORA framework.

It is expected that the total number of unique genome k-mers in $G'$ will grow sublinearly as new individuals are added.

**Practical observation of sublinear scaling**

As the theory described above suggests, the CORA framework achieves sublinear scaling for the coarse mapping stage. In the absence of sequencing errors, the total number of read sequences that are processed in the coarse mapping stage is in effect constant after about 100 million 2×100bp reads (Figure 2-5).

In order to illustrate the sublinear-scaling property of the CORA framework in practice with sequencing errors present, we computed the rate of redundancy within...
simulated read datasets with various sequencing error rates and read depth-coverage (Figure 2-6). The rate of redundancy is calculated as the total number of $k$-mer sequences sampled from the reads divided by number of unique $k$-mers processed by CORA during the 'coarse mapping' stage. Results indicate that the rate of redundancy monotonically increases, even in the presence of sequencing errors. Moreover, the rate of increase in redundancy often alternates between slow and fast as we increase read depth coverage, which is supported by the saturation rate of various Hamming shell radii as described above. Across different sequencing error rates, the rate of increase is slow universally for lower coverage datasets (below 1 million reads), since $k$-mers have not yet fully covered the reference sequence. As read depth-coverage increases further, more $k$-mers overlap with the previously encountered reads, resulting in a faster increase in the rate of redundancy, thus allowing CORA’s compression scheme to produce fewer $k$-mers to be processed by the coarse mapping stage. This feature can be observed on all E2, E0.5, and E0.125 datasets between 10 million and 100 million reads. Depending on the sequencing error rate, the rate of redundancy may flatten again at a higher depth-coverage level (such as between 100 million and 1 billion reads in E0.125 and roughly 100 million reads in E0.5). This observation is due to the fact that as rate of redundancy increases, the non-redundant $k$-mer set becomes dominated by $k$-mers containing sequencing errors which are less likely to be encountered multiple times. Ultimately, with enough depth-coverage, $k$-mers containing sequencing errors also become redundant, and the rate of increase in redundancy becomes steep again (as observed around 1 billion reads in E0.5).

**Implementing read collapsing and coarse mapping**

We implemented read collapsing with an unordered hash table (unordered map class in C++ standard library), consisting of DNA sequences (the lexicographically smaller of the forward sequence or its reverse complement) as hash keys and a string that represents the encoded collapsed read names as the value for each key.

For large datasets with many unique sequences, which would cause the hash table to exceed the memory limit, we use a multi-pass approach to keep the size of the
Figure 2-6: Sublinearity analysis of CORA framework on simulated paired-end read data from hg19 chromosome 20 with 0.1% mutation rate and 2%, 0.5%, and 0.125% sequencing errors (E2, E0.5, and E0.125), respectively. Rate of redundancy is calculated as the total number of k-mers in the read dataset divided by the number of k-mers processed by CORA during the coarse mapping stage. Results indicate that rate of redundancy monotonically increases even in the presence of sequencing errors; therefore, for all three datasets the CORA framework spends less coarse mapping time per additional read, indicating sublinear scalability regardless of sequencing errors in practice.
Table 2.2: Runtime and memory cost of CORA’s collapsing stage for the FIN1 dataset with 54bp k-mer length for collapsing schemes with different number of passes.

<table>
<thead>
<tr>
<th>Number of passes</th>
<th>Peak memory use (GB)</th>
<th>Total runtime for collapsing (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>54.19</td>
<td>5464.97</td>
</tr>
<tr>
<td>18</td>
<td>39.25</td>
<td>4783.58</td>
</tr>
<tr>
<td>24</td>
<td>31.61</td>
<td>4592.06</td>
</tr>
<tr>
<td>32</td>
<td>25.87</td>
<td>4565.42</td>
</tr>
<tr>
<td>40</td>
<td>23.39</td>
<td>4604.68</td>
</tr>
<tr>
<td>48</td>
<td>19.68</td>
<td>4635.77</td>
</tr>
<tr>
<td>96</td>
<td>19.68</td>
<td>5288.44</td>
</tr>
</tbody>
</table>

In-memory hash table within specified memory bounds. This approach is akin to the multi-pass one we used in order to compute the exact homology table. The amount of memory used during the collapsing stage is directly dependent on the number of passes performed within the multi-pass hashing scheme.

Below we demonstrate the memory/runtime trade-off of CORA’s collapsing stage for the human FIN1 dataset from the 1000 Genomes Project with paired-end 108bp reads and 54bp k-mer length, using a 4-nucleotide splitting scheme (e.g. AA – CT), ranging from 12 to 96 passes (Table 2.2).

Memory/runtime trade-off results indicate that memory use for the collapsing stage can be reduced by increasing the number of passes. Note that for the 4-nucleotide splitting scheme, there is no further memory reduction after 48 passes. However, the memory can be further reduced by using a 6-nucleotide splitting scheme instead (e.g., ATA-GGT with 3 nucleotides at each end of the k-mer), resulting in a slightly higher total runtime cost. In terms of runtime, too few passes are not desirable due to the incremental cost of hash-table operations for larger tables. On the other hand, too many passes are not desirable either, due to the overhead cost of I/O handling. We show that roughly 32 passes is ideal in terms of the runtime optimization for human datasets, resulting in less than 26GB memory use. However, the CORA software allows users to increase the number of collapsing stage passes in order to meet their runtime or memory use preferences.

As the CORA framework utilizes an off-the-shelf aligner (mapper) for its coarse mapping stage, the memory cost is directly dependent on memory needs of the aligner.
used. In the cases of BWA and Bowtie2, the memory cost of coarse mapping is dominated by the k-mer collapsing stage. Since the link construction stage involves scanning only a SAM file to generate the read links, its memory consumption is negligible.

2.2.3 Homology table traversal

The main goal of this phase is to infer final paired-end read mapping results, making use of the precomputed homology table of the reference, as well as the compact links table representing the read dataset.

As links and homology tables are all represented in terms of positions in the reference and the differences from it, rather than the read sequence itself, almost all of the computation in this phase of the CORA framework is performed in terms of edit operations rather than direct sequence comparison. Since a link generally has much fewer edits than the number of bases in the read, utilizing algebraic relations within this 'edit space' results in a much more efficient sequence comparison method than the base-to-base comparison approach used in conventional read mapping.

A schematic representation of the homology table traversal for a single read link is provided in (Figure 2-7). Each link connects one or more k-mers to a position in the reference (the anchor of the link) in forward or reverse complement direction, indicating homology within a small number of edits. This anchor is either a unique k-mer in the reference genome or associated with an equivalence class within the exact homology table. In the latter case, the anchor is linked to the equivalence class in the forward or reverse complement direction, with a block offset value indicating the starting location (say offset = \(p\)) of a substring of the equivalence class representative that is identical to the anchor sequence. If an anchor does not belong to any existing equivalence class, it is considered to be the representative of its own single-item equivalence class.

A representative of an equivalence class points to all members of the equivalence class for each valid offset (until the end of the block) in the forward or reverse complement direction as described above. Among these pointers, only the ones with the
Figure 2-7: Homology table traversal scheme of CORA framework. Each read link, representing one or more reads, points to the genomic location determined by the coarse mapping stage in either forward or reverse complement direction, indicating homology within a small number of edits. This locus is either unique in the reference genome or associated with an equivalence class in the exact homology table. In the latter case, the anchor is linked to the equivalence class in the forward or reverse complement direction, with a block offset value indicating the starting location of a substring of the equivalence class representative that is identical to the coarse genome target. The representative of the equivalence class points to all members of the equivalence class for each valid offset (until the end of the block) in forward or reverse complement direction. Furthermore, the equivalence class representative can be linked to other equivalence classes through the inexact homology table. Each of these pointers contains the direction of homology (forward or reverse complementary), block offset in the compressed inexact homology block representation, and the edit script to convert one class representative to another. Provided that the juxtaposition of the edits from a read link to the anchor and from the anchor’s class representative to a neighboring equivalence class still contains less than or equal to the user specified number of errors, read mappings associated with all members of these neighbor equivalence classes will also be reported.
offset $p$ are relevant to the read link, and solely for these will the inferred mapping result be reported. Furthermore, the equivalence class representative of the anchor can be linked to other class representatives through the inexact homology table. Each of these pointers contains the direction of homology (forward or reverse complement), block offset in the compressed inexact homology block representation, and the edit script to convert one $k$-mer to another. Provided that the juxtaposition of the edits from a read link to the anchor and from the anchor’s class representative to the neighboring equivalence class still contains less than or equal to the Hamming distance threshold, read mappings associated with all members of this neighbor equivalence class will be reported.

For single $k$-mer reads (e.g., $\leq 64$ bp) that are collapsed as a whole read into read links, the final mapping output can be generated by the procedure described above. For paired-end and/or longer reads that contain multiple $k$-mers, all links relevant to the read are traversed to generate the final mapping(s). CORA achieves this traversal by loading the links table in multiple passes to avoid exceeding the memory limit. These links are then independently converted into mappings. In the end, these mappings are merged in order to generate the final set of paired-end mappings.

**Traversing with indels**

Even if the homology table data structure is built using a Hamming distance metric that only allows substitutions, it is possible to report gapped final mappings with indels (insertions and deletions), given that the coarse mapping is performed using an off-the-shelf tool that can perform gapped mapping and the error penalty assigned to insertions and deletions is greater than or equal to the penalty assigned to substitutions (e.g., Levenshtein distance that assigns identical penalties for substitutions, insertions, and deletions).

The homology table is initially built to provide Hamming neighbors of a $k$-mer in the reference for a fixed error distance threshold, $E$. However, it is also possible to recover homologies of $(k - e)$-mers in the reference ($e \leq E$), which is guaranteed to discover all homologies within a Hamming error distance threshold of $E - e$, ignoring
the matches/mismatches in the last e bases of homology table k-mers. Since e is often much smaller than k, most of the E-distance homologies of (k - e)-mers will also be recovered. This ability to search for (k-e)-mers in the homology table enables neighborhood search of k-mers that are coarse mapped with insertions in them. In the case that a k-mer is coarse mapped with a single base insertion in the k-mer link (which corresponds to a single base deletion from the reference), the k-mer is aligned to a (k - 1)-mer in the reference. Using the above approach, CORA can recover most of the homologous regions of the (k - 1)-mer.

In the case of a single nucleotide deletion in the k-mer link (insertion in the reference), the k-mer corresponds to a (k + 1)-mer in the reference. In this case, the homology table can enumerate the k-mer homologies for any k-mer contained within this (k + 1)-mer and then determine whether the (k + 1)-mer homology is within the valid error distance threshold by checking the similarity of the last base in the (k + 1)-mer.

For multiple insertions and deletions in the k-mer link, the homology table traversal strategy is determined by \( t = \#\text{deletions} - \#\text{insertions} \). If \( t < 0 \), the first strategy employed ignores the last \( t \) bases in the k-mer homologies; if \( t > 0 \), the second strategy employed checks for \( t \) additional bases at the end. If \( t = 0 \), the initial version of the k-mer based homology table traversal is performed, while handling shifted bases due to insertions and deletions within the k-mer when juxtaposing k-mer homology edits with the edits from the k-mer link. The \( t \) value is also important when merging multiple k-mer link mappings into a whole read during the final mapping stage. If one of these k-mers have a positive or negative \( t \) value, k-mer merging is performed between accordingly adjusted mapping positions.

**Mapping recovery to ensure high sensitivity**

Although the links table is a lossless representation of the read sequences in the original dataset, there are three main reasons why a particular read mapping may be missed during standard traversal of the homology table:

1. Using an inexact homology table with a distance threshold lower than twice the
mapping error.

2. Loss of sensitivity during coarse mapping due to imperfect sensitivity of the coarse mapper.

3. Non-uniform distribution of errors among k-mers sampled from the read during collapsing.

In order to achieve near-perfect sensitivity, CORA employs a recovery scheme which recovers a portion of the missed mappings.

In order to recover mapping locations of reads with unmapped k-mers during coarse mapping, CORA keeps track of all k-mers that are unmapped during the coarse mapping stage. Then the homology table traversal generates all mapping loci for the other k-mers of the read. If more than a certain percentage (e.g., 75%) of its k-mers constitute a viable mapping (i.e., only missing the unmapped k-mers but satisfy the error distance and min/max read insert length conditions for the others), each unmapped k-mer is extended using a base-by-base comparison method. In the case of gapped mapping base-by-base comparison is replaced by a banded dynamic programming algorithm in order to detect indels.

As the mapping recovery stage is performed independently from homology table traversal, the dynamic programming algorithm within this stage can handle larger error distances than the homology table allows for (e.g. 10 or more mismatches), as well as affine gap penalties to allow for dynamic trimming of reads (when the k-mer is at the end) or alternatively capture indels longer than the error distance threshold.

When constructing a homology table, in order to guarantee perfect sensitivity for obtaining all matches of a k-mer in the reference genome within a Hamming distance of D, the homology table should ideally use a distance threshold of 2D. This is due to the fact that, even when a read k-mer is within distance D of both the coarse mapping locus and final mapping locus, the distance between the final mapping locus and the coarse mapping locus could be as large as 2D, if the set of edits from the read to the coarse mapping locus and the read to the final mapping locus are disjoint. However, as homology tables with large error rates take substantially
longer to construct and consume larger memory, for practical performance reasons, the threshold of the homology table can be chosen to be less than $2D$ (such as $D$, which is used for the results in Figures 2-1 and 2-9), resulting in imperfect sensitivity. In order to recover mappings that have distance higher than $D$ to the coarse mapping, CORA again employs a nucleotide-level string comparison given that a large percentage of the remaining $k$-mers constitute a viable mapping.

In subsection 2.3.2, we provide an experimental analysis on the runtime and sensitivity trade-off of the recovery scheme for a variety of homology table distance thresholds and perform a comparative analysis on experiments when the recovery scheme is enabled and disabled.

**Implementing homology table traversal**

For the human (hg19) read-mapping experiments presented with a $k$-mer length of 54bp, the exact homology table requires ~0.8 GB of memory, whereas the inexact homology table that represents all reference homologies of Hamming distance 2 requires ~22.2GB of space in memory. For CORA’s fast best-mapping mode, only the exact homology table is required, whereas for all-mapping mode or a more sensitive best-mapping mode, CORA requires both exact and inexact homology tables to be represented in the memory. Apart from the homology table, CORA also utilizes lookup tables for the reference in order to identify the nucleotide at each locus in the reference as well as corresponding indices in the homology tables. This table requires an additional 2.4 GB of memory for best-mapping modes that do not utilize the inexact homology table and 24.8 GB for other mapping modes (the higher memory usage is due to index values needed for the inexact homology table). Among the best-mapping modes, when the inexact homology table is used, both sensitivity and runtime is slightly higher than the modes that do not utilize the inexact homology table (Figure 2-9).

Finally CORA allocates an adjustable amount of memory to represent the read links in RAM for fast access. For the FIN1 dataset, the maximum memory size needed for allocation is 2.76 GB (as opposed to 29.1 GB, which is the original FASTQ file
Table 2.3: Runtime and memory cost of CORA’s homology table traversal stage on FINI dataset for best and all-mapping for different memoization thresholds.

<table>
<thead>
<tr>
<th>Memoization Threshold</th>
<th>No memoization</th>
<th>20</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Mode</td>
<td></td>
<td>1331</td>
<td>1200</td>
<td>1125</td>
</tr>
<tr>
<td>Best-mapping (Fast mode)</td>
<td>Runtime (seconds)</td>
<td>11260</td>
<td>7743</td>
<td>7797</td>
</tr>
<tr>
<td></td>
<td>Peak memory</td>
<td>48 GB</td>
<td>51 GB</td>
<td>54 GB</td>
</tr>
</tbody>
</table>

size). However, this amount can be reduced by a factor of X through a multi-pass option that scans the read link file X times.

For speed optimization on high-coverage read datasets, CORA uses a memoization scheme. Memoization is a widely-used optimization technique in computer science that aims to gain speed through storing the output of expensive function calls. For the read links that represent more than a certain number of k-mers in compressed form, CORA traverses the homology table once and saves the inferred genomic positions for the link in a lookup table. Then for each of the k-mers within the link, this lookup table is used instead of re-computing the genomic neighbors using the homology table. In whole genome real-data experiments (Figures 2-1 and 2-9), we chose this threshold to be 20 which resulted in a good balance between runtime improvement and additional memory required for the lookup table. Below we present a breakdown of runtime and peak memory usage (maximum resident set size) of homology table traversal for best- and all-mapping, as well as a variety of memoization threshold values. While CORA’s memoization scheme results in moderate speed gains for best-mapping with a negligible increase in peak memory usage, it provides drastic speed gains for all-mapping (when the threshold is set to 10 or 20) with a relatively small increase in peak memory use (Table 2.3). All experiments reported on human datasets (Figures 2-1 and 2-9) with memoization threshold 20 can be run on a machine with ~64GB of free RAM.
Complexity of seed-and-extend with homology table

We now show that pairing a seed-and-extend based mapping method with a homology table reduces the asymptotic complexity of mapping on average by a factor of the redundancy of the reference genome with respect to the read sequence. Note that this complexity analysis is independent of the size of the read dataset or the redundancy within. As shown earlier, in the case of coarse mapping in CORA, the asymptotic complexity is further reduced by a factor of the redundancy within the read dataset itself due to k-mer compression of reads. Furthermore, even though the measure of redundancy of the genome with respect to the read is correlated with the redundancy within the reference genome, the former is lower in cases such as uniquely mapped or unmappable reads as we quantify later.

Though we initially assume an ungapped all-mapping model, we extend this analysis later to the gapped mapping scenario. The complexity analysis also holds in the best-mapping version of the problem, except a corner case which we elaborate on later. We always consider the perfect sensitivity versions of the problem in our complexity analysis, and therefore exclude heuristic or greedy algorithm based mappers that generate imperfect sensitivity mapping results. We also assume a strict seed-and-extend mapping model, where each seed is separately searched in the reference genome, without any precomputation on the read dataset (such as CORA’s k-mer compression scheme) and all different seed hit loci are extended.

For our analysis below, we need to define a mapping method that reports an arbitrary mapping locus in the reference genome within a specified distance threshold of the read without any preference for its locus or actual distance, or none if there is no mapping locus within the distance threshold. We will call this an ‘any-mapper’. For the following analysis, we will assume that we will be pairing a seed-and-extend based any-mapper with the homology table in order to build an all-mapper, and compare its runtime complexity to a seed-and-extend based all-mapper. We will assume that the underlying complexity of seed-and-extend operations are consistent across the mappers.
We define the seed operation as sampling a shorter substring from the read and mapping it exactly (i.e. with no errors) to all matching locations in the reference genome which then become candidate loci for the extend stage. The extend stage is defined as identifying which of the candidate loci are valid mappings within the similarity threshold (in the case of all-mapping) or identifying the locus with the highest similarity score (in the case of best-mapping). To achieve this, the seeds are extended towards either direction of the reference match.

For ease of analysis, we assume that the read is single-end and is of fixed length $k$. We will refer synonymously to a read and the $k$-mer it represents from here on.

Let us define $R(q, d) = \{ p \in G : \delta(p, q) \leq d \}$, the local $k$-mer neighborhood in our genome $G$ of the query $k$-mer $q$ for Hamming distance $d > 0$. Note that $|R(q, d)|$ serves also as a good measure of the redundancy of the reference genome with respect to the read $q$. This aligns with our natural intuition for redundancy as describing the self-similarity of $G$ with respect to $q$, as $|R(q, d)|$ is also the output size of a perfectly sensitive all-mapper that maps $q$ to $G$ with a distance threshold of $d$. We exclude unmappable reads from this complexity analysis as $|R(q, d)| = 0$; however, since the use of an any-mapper or a homology table in the case of unmappable reads will not have an effect on the runtime, its asymptotic complexity behavior is similar to the uniquely aligned reads with $|R(q, d)| = 1$.

Consider an all-mapper $\tau$ employing the seed-and-extend strategy as follows to return $R(q, d)$ with guaranteed perfect sensitivity:

1. Break the read $q$ of length $k$ up into length $s$ seeds.

2. Find all locations in the genome that match these seeds in order to enumerate tentative mapping locations.

3. Extend each seed hit to check if the tentative mapping location is within distance $d$ of $q$.

4. Report positions of the mapping locations within distance $d$. 
Note that the read \( q \) must be broken up into at least \( d + 1 \) seeds to ensure that at least one seed will have a perfect match to the reference index \([59]\). Thus, a seed length \( s \leq \lfloor k/(d + 1) \rfloor \) is required in order to guarantee perfect sensitivity assuming uniform seed lengths.

Let us define the complexity function of this seed search operation (2) of the seed-and-extend based read mapper to be \( f(s, |G|) \). When considering all of the seed search operations, the complexity of the seed stage overall will be \( O(d \times f(s, |G|)) \). If we assume that the data structure used will allow accessing each additional seed hit in the genome in constant time, the time required to access all additional seeds will be dominated by the extend stage time, thus we can ignore it.

The computational cost in step (3) is finding whether each seed match in the reference is a valid read mapping with at most \( d \) mismatches. Let us define the complexity function of this extend operation to be \( g(k, |G|, d) \). In the case of a hash table implementation, the runtime complexity of the extend operation will be \( O(k) \) as we need to compare \( k - s \) bases between the read and the reference locus. As we need to perform the extend stage for each seed hit, the total complexity of this stage is \( O(\text{number of seed matches} \times g(k, |G|, d)) \). The cost of reporting the mapping positions will be dominated by number of seed matches, thus we can ignore it for the all-mapper.

Now let us consider pairing an any-mapper \( m \) with a homology table \( H \) to build an all-mapper. Given a query string \( q \), \( m(q) \) returns either the location of some substring of \( G \) within distance \( d \) of \( q \), or that there is no such location. We construct the any-mapper similarly to the seed-and-extend all-mapper above, except that we short-circuit the extend stage once a location within distance \( d \) of \( q \) is found. In expectation, the runtime of this any-mapper will be \( O((\text{number of seed matches} \times g(k, |G|, d))/|R(q, d)|) \), assuming we process the seed hits in a randomized order.

Now, let us consider the computational cost of using a homology table to generate all of the mappings of \( q \) to the reference within distance \( d \), given \( m(q) \) from the any-mapper. Below, we redefine the homology table in a slightly different way than the description in 2.2.1 Homology table of the reference. This change is due to
the fact that, here we are concerned solely about the asymptotic complexity of the problem rather than practical concerns when building the CORA framework, such as memory/disk-space or preprocessing time costs. For example, we exclude here the block-merging step of the actual homology table used in the CORA framework, which in practice saves a lot space with little or no effect on real runtime, but might increase the worst case complexity of search.

We will construct the homology table for distance $2d$ as follows:

1. Bin the locations in the genome together by exact $k$-mer matches. Because each bin contains all locations with a particular $k$-mer, we will refer synonymously to the bin and the $k$-mer it represents.

2. For every pair of bins $b_1$ and $b_2$, create a link between them if the distance between their $k$-mers is $\leq 2d$. In that link, store the position of the mismatch(es).

Then our homology-table augmented all-mapper algorithm will be as follows:

1. Use the any-mapper to get $m(q)$, which is within distance $d$ of $q$.

2. Look up $m(q)$ in the homology table.

3. In the homology table, for every link from $m(q)$'s bin, check if following that link will lead to a bin with $k$-mers within distance $d$ of $q$.

4. Traverse the appropriate links and return $R(q,d)$.

By the triangle inequality, we are guaranteed to get all of $R(q,d)$ using this scheme, so long as the homology table has all links with distance $\leq 2d$. Note that because homology table links store the positions of the mismatches, we need only check the mismatches to determine the Hamming distance between the new bin and $q$. Thus, homology table traversal takes $O(d \times |R(m(q), 2d)|)$ time. Total runtime for the homology-table augmented all-mapper is thus $O(d \times f(s, |G|) + (\text{ #seed matches } \times g(k, |G|, d)) / |R(q,d)| + d \times |R(m(q), 2d)|)$.

This complexity can be further reduced if we modify the homology table to have a precomputed mismatch table for each bin, that specifies which other bins will be
within distance $d$, depending on the locations of differences between $m(q)$ and $q$. For example, a bin can point towards different sets of neighboring bins, in the case that the $5^{th}$ position in the $k$-mer has a mismatching base ‘G’ versus the $26^{th}$ position has a mismatching base ‘T’. In the naive version, this would require us to have a table of size $C(k, d) \times 3^d$ for each bin, even though more space efficient data structures are possible that exploit the sparsity of this table. With this expanded homology table, total runtime complexity reduces to $O(d \times f(s, |G|) + (\text{number of seed matches} \times g(k, |G|, d))/|R(q, d)| + |R(q, d)|)$, assuming output costs consist of reporting only the position in the genome in constant time for each mapping.

The initial complexity of the all-mapper is $O(d \times f(s, |G|) + \#\text{seed matches} \times g(k, |G|, d))$. We did not add an additional term for the output cost $|R(q, d)|$ as number of seed matches $= \Omega(|R(q, d)|)$. If $d \times f(s, |G|) = O(g(k, |G|, d))$ as well as $|R(q, d)|^2 = O(\text{number of seed matches} \times g(k, |G|, d))$, we can claim that an any-mapper paired with the homology table is faster than an all-mapper by a factor of $|R(q, d)|$, which corresponds to the redundancy of the reference with respect to the read. Assuming that $f(s, |G|) = O(s = [k/(d + 1)])$ and $g(k, |G|, d) = \Omega(k)$, the first condition is satisfied. Both of these assumptions are valid, since a hash-table is able to satisfy the first condition; and since we assume no pre-computation on the read dataset, the remaining bases in the read, apart from the seed loci, and error positions are unknown prior to the extend stage, thus requiring $\Omega(k)$ operations for a $k$-mer with a seed hit.

For the second condition, we need to show that $\text{number of seed matches} = \Omega(|R(q, d)|^2)$. In the generalized case, where we have $d + 1$ seeds of length $s = [k/(d + 1)]$, the maximum number of unique $d$-away $k$-mers we can construct from $q$ is $C(k, d) \times 3^d$. Whereas the total number of possible unique $k$-mers that contain at least one exact seed in their corresponding $k$-mer position is $4^k - (4^{[k/(d+1)]} - 1)^{d+1} = \Omega(4^{k/2})$. If the reference is assumed to be constructed in a non-adversarial way by selecting $k$-mers from a unique $k$-mer list with repetitions, by expectation the second type of $k$-mers will appear exponentially more often than the first type of $k$-mers, for $0 < d < k/2$. In this model, it is easy to show that the number of seed hits
grows faster than quadratic in the number of valid mappings, satisfying our second condition.

By using the homology table, if the runtime is dominated by the number of seed matches, we are able to get acceleration proportional to the local redundancy of our query in the genome. Amortized across all reads and their seeds, this runtime will by expectation correspond to the average redundancy of the reference genome with respect to the read dataset. This quantity is a function of the similarity between the donor genome and the reference genome as well as the sequencing error rate.

In the perfect sensitivity version of the best-mapping problem, in which a mapping with minimal distance to the reference genome is guaranteed to be found provided that one exists within the predetermined distance threshold $d$, the seed stage will be performed as earlier for both best-mapper and any-mapper. The extend stage will also still be the same for an any-mapper. If we make sure that the precomputed mismatch table in the homology table points to the interval of minimal distance mapping, the traversal stage can find a random best-mapping in constant time. In the case that there are no perfect matches, the complexity of the extend stage will not be changed as each seed hit needs to be extended to make sure that there is not a mapping with a lower number of errors. In the case that there is no more than a constant number of perfect matches for the read in the reference, the complexity of the extend stage for perfect sensitivity best-mapping will be the same as perfect sensitivity all-mapping, as the expected number of seed hits that need to be extended is not changed. Even in the presence of relatively frequent perfect matches, the any-mapper will be faster, but not necessarily as much as a factor of $|R(q,d)|$.

In the gapped mapping version of the problem the extend stage will be more costly due to the detection of insertions and deletions while extending. However, the seed stage will stay the same as we are still guaranteed to find all read mappings if $d+1$ disjoint seeds are sampled from the read of length $s \leq \lfloor k/(d+1) \rfloor$. Since the cost of the gapped extend stage is lower bounded by the ungapped extend stage, our analysis still holds in the gapped all-mapping and gapped perfect sensitivity best-mapping problems.
Note that for uniquely aligned reads as well as unmappable reads, we do not expect any speedup of more than a constant factor. However, even in these cases compressive acceleration is possible due to \(k\)-mer compression of reads in the CORA framework.

Moreover, in our asymptotic analysis in this section, we did not include mapping methods based on greedy algorithms that only check a subset of the seed hits for mapping (in particular best-mapping), resulting in imperfect sensitivity mapping results. Thus, other practically useful speed improvements of methods that cannot guarantee perfect sensitivity (e.g. Bowtie2, BWA, Masai, GEM, etc.) are left outside of this complexity analysis.

### 2.2.4 Comparison to other compressive frameworks

Here we describe existing work and how each differs from CORA’s compressive read-mapping framework.

In their pioneering work, Veeneman et al. [5] describe a read-mapping methodology, Oculus, which wraps an existing aligner and aims to accelerate read mapping via elimination of duplicate reads in the input read dataset prior to mapping; the method reconstructs final mappings by regenerating additional mapping results for the unmapped, eliminated reads using results from their mapped copies. In essence this approach is similar to a primitive version of CORA’s single-end read-mapping without using a homology table, sampling only a single whole read for the collapsing stage. Though a simplistic approach like this can provide modest compressive acceleration for highly redundant read datasets that are single-end and have short read-length, it strikingly fails to provide any meaningful acceleration for longer read-length paired-end read datasets, such as the \(2 \times 108\)bp read datasets from 1000 Genomes Project used in FIN1, FIN2, FIN4 experiments presented. In contrast, CORA can provide orders of magnitude speed increase over existing aligners on the FIN4 dataset, whereas the total number of duplicate paired-end reads in this dataset is less than 3\%, which represents a hard upper bound on the acceleration Oculus can provide. Moreover, Oculus requires time to de-duplicate the input and expand the output mappings for
eliminated reads, resulting in even smaller margins for acceleration, which is roughly 2%. As opposed to Oculus, CORA’s acceleration stems from the fact that $k$-mers shorter than read length are used for compression (with exponentially increased odds for duplicates), splitting a longer read into multiple smaller chunks and constructing the whole paired-end read mapping results after coarse mapping through the use of the homology table rapidly enumerating multiple mapping positions for each $k$-mer in the reference genome. Without the use of a homology table, such a $k$-mer based compressive mapping approach would be prohibitively costly.

Apart from identifying exact duplicates, Mahmud and Schliep explored wrapping existing aligners using a whole-read clustering scheme (TreQ-CG) where reads that are similar within a distance threshold are clustered together [6]. While some level of acceleration is gained from this approach, it comes with a significant loss in mapping sensitivity. Furthermore, whole-read inexact clustering is a costly solution, causing CORA to be able to run its entire mapping pipeline several times over before TreQ-CG completes even its initial read clustering stage.

In addition to end-to-end compression of reads through de-duplication or clustering, some read-mappers [47, 48, 49] have explored the idea of jointly representing seeds or putting them in a compact data structure, in order to accelerate the seed-matching stage of mapping. Below we summarize the idea of clustering seeds prior to seed-matching step and describe the conceptual and practical differences between the CORA framework and joint seed-matching approaches.

Inspired by the efficient seed-and-extend alignment algorithm of BLAST [13], almost all short-read mapping algorithms are designed with two main computational stages: the seed stage and the extend stage. The seed stage can be broadly described as sampling a short subsequence (or multiple subsequences) of the read and matching it exactly (or with very high similarity) to multiple tentative locations in the reference genome which then become candidate loci for the extend stage. The main computational cost of the seed stage is searching for seed sequences in a data structure that represents the reference genome (e.g. a hash table or suffix array). The extend stage can be broadly described as identifying which of the candidate loci are valid mappings.
within the similarity threshold (all-mapping) or identifying the locus with the highest similarity score (best-mapping) through extending the seeds towards either direction of the reference match (as well as filling in gaps in the case of spaced seeds). The main computational cost of the extend stage is to perform similarity comparisons between the reference and the bases in the reads that are not contained in the seed, which quite often corresponds to the majority of the bases in the read, as well as the majority of the computational cost of read-mapping. Most read mapping methods already employ a preprocessed reference index that allows fast identification of seed matches in the reference genome, thereby speeding-up the seed stage of mapping. Furthermore, some methods such as mrsFAST-Ultra, Masai and BWA-SW employ a seed clustering scheme that also process the seed-set on-the-fly in order to reduce redundancy in seed-matching comparisons across multiple seeds from different reads, which aim to further accelerate seed matching stage.

Conceptually, there are three key differences between CORA's k-mer read compression scheme and joint seed-matching schemes that exist in the literature. Firstly, the length of the k-mers that CORA collapses are chosen to be significantly longer (40-60bp) than the seed lengths that are typically used by read-mappers (8-20bp). Therefore, the type of redundancy that CORA leverages is based on longer homologous regions across both the reference genome and the reads sequenced from these regions across multiple individuals; on the other hand, existing methods rely on a large number of spurious hits that short seeds would normally produce, often increasing the computational cost of the extend stage. Longer k-mer lengths are essential to the CORA framework, as CORA's k-mer read compression scheme is complemented by a homology table that enables very efficient all-mapping via rapid enumeration of homologous regions in the reference genome. Building a homology table with such short seeds would be intractable, as it would be infeasible to pre-process, store and enumerate the exponentially larger number of homologies. Secondly, by joint seed-matching, read mappers aim to accelerate only the seed stage of mapping (which requires relatively less costly computations in the form of data structure access operations compared to costly nucleotide-level comparisons of the extend stage), whereas
CORA’s k-mer read compression accelerates the entirety of the coarse mapping stage, which includes both the seed stage and the costlier extend stage. Thirdly, while the seed-matching stage requires identifying many regions in the reference genome in order to produce candidate mapping loci for the extend stage (otherwise the sensitivity of mapping would suffer), it suffices for CORA’s coarse mapping stage to only produce any single region in the reference genome, that is, within a specified distance of the compressed k-mer. As we analyzed above, the asymptotic complexity of this search operation is lower than seed-and-extend based inexact mapping by a factor of the redundancy in the reference genome with respect to the reads.

In practice CORA’s runtime performance is superior to mrsFAST-Ultra with near-perfect sensitivity and to Masai with much better sensitivity. BWA-SW is primarily designed to efficiently map longer reads (>200bp) and costlier than other BWA methods for FIN1-FIN4 datasets.

Overall CORA’s k-mer based read compression, complemented by its compressive homology table, represents a novel methodological advance in read mapping. This advance is also apparent in the superior practical performance, in terms of both speed and sensitivity, of our implementation of the CORA compressive read-mapping acceleration framework as compared to existing tools.

2.2.5 Software features

We provide an implementation of CORA at http://cora.csail.mit.edu, which can readily perform rapid gapped read-mapping for paired-end Illumina read datasets, given a FASTA/FASTQ input file, reporting a SAM format mapping output file. A full overview of the CORA framework is given in Figure 2-8. We discuss features of our implementation below.

We implemented the CORA framework in the C++ programing language. It does not have any dependencies apart from the user-specified coarse mapping tool to be incorporated into the framework. Virtually all short-read mapping tools can be plugged into the CORA framework with minimal or no changes to the implementation. Currently integrated tools and mapping modes include BWA [40] aln and mem, 77
**Preprocessing Stage:** Homology table generation and reference indexing

**Processing Stage 1:** K-mer collapsing
- Input: Reference sequence (FASTA), k-mer length value $K$, distance threshold value $S$
- Output: List of unique k-mers
- Auxiliary read look-up table
- Perfect k-mer links

**Processing Stage 2:** Coarse mapping
- Input: Reference sequence (FASTA), Reference index
- Output: Coarse mappings (SAM)
- Link construction
- K-mer links

**Processing Stage 3:** Homology table traversal
- Input: Homology table traversal
- Output: Mapping output (SAM)

---

**Figure 2-8:** An overview of CORA framework and input/output relations between different stages of the CORA software. Green, purple, yellow and red panels describe pre-processing, k-mer collapsing, coarse mapping and homology table traversal stages of the CORA pipeline respectively. The grey panel on the lower right provides a legend for different types of boxes and arrows used in the diagram. **Green panel:** The inputs for the preprocessing stage are a reference sequence in multi-FASTA format, k-mer length value $K$, distance threshold value $S$, and an executable binary for an off-the-shelf read mapper to be used in coarse-mapping. The off-the-shelf mapper’s indexing algorithm is called in order to generate a reference index. Exact and inexact homology tables are constructed and compressed to be used in the homology table traversal stage. **Purple panel:** K-mer collapsing stage takes in a set of FASTQ files together with the k-mer length value as input and generate a list of unique k-mers (with compact IDs that encode reads that contain the extracted k-mers).
Figure 2-8: (cont.) An auxiliary lookup table is generated for some of the k-mer IDs that are known to cause problems during coarse-mapping (e.g. very long IDs). Optionally, k-mer collapsing stage uses the reference genome in order to identify k-mers that are identical to a k-mer in the reference, in which case these k-mers will be separately reported as perfect k-mer links to the reference genome skipping the coarse mapping stage. **Yellow panel:** The coarse mapping stage requires an off-the-shelf mapper executable (which could be built-in tools such as BWA-aln/BWA-mem, Bowtie/Bowtie2, mrsFAST/mrsFAST-Ultra but could also be manually described through manual mode) and its reference index in order to place each of the unique k-mers to a locus in the reference. Afterwards the coarse-mappings are converted to k-mer links within the link construction step of CORA. In the case that k-mer collapsing was performed with the reference sequence, the perfect k-mer links from the previous stage are merged with the k-mer links generated from the link construction stage. **Red panel:** Homology table traversal stage takes in the k-mer links from the previous stage, reference sequence and the exact/inexact homology tables, in order to generate the final set of mapping (in SAM format) by traversing the homology table(s) following the k-mer links. While some mapping modes can be performed only using the exact homology table (e.g. fast best-mapping mode), other mapping modes would require the inexact homology table as well. Optionally the original read dataset is used in this stage, in order to print the SAM file with the original read names (as opposed to just their order information in the input file) and/or the quality scores.

Bowtie [60], mrsFAST [42], and mrsFAST-Ultra [47].

The current implementation of CORA can perform mappings of paired-end read datasets with uniform read-lengths (between 2 x 36bp and 2 x 150bp) within a Hamming distance threshold. In addition, users specify an allowable insert length interval for mate-pairs. CORA performs end-to-end mapping of reads, which corresponds to the global alignment of each read to a locus in the reference. The current implementation of CORA does not make use of quality scores within the alignment (all substitutions or indels have equal weight within the Hamming/Levenshtein distance metric); however, quality scores can optionally be printed for downstream use in a sequence analysis pipeline.

CORA allows the user to specify the distance metric used for mapping, Hamming or Levenshtein (edit) distance, without any modification needed to the homology table. Current version of CORA software allows the user to set the distance threshold up to 6 edits/substitutions all and best-mapping. For best-mapping, the CORA package has available mapping inference modes that can also handle distances higher than 6.
CORA allows the user to specify to output all-mappings, best-mappings, unique-mappings or best stratum mappings, the latter of which corresponds to outputting all mappings within the highest tier. Furthermore, CORA enables reporting of the original read names and quality scores from the input read dataset, or alternatively assigns numbers to the reads that represent their order in the read dataset.

2.3 Results

2.3.1 Datasets used

Real NGS datasets used

For the experimental results shown (Figures 2-1 and 2-9)), we used three real-life whole-genome sequencing datasets (mentioned below as FIN1, FIN2 and FIN4) with varying depth-coverage obtained from the 1000 Genomes Project [61]. These datasets include Illumina sequencing reads from 7 Finnish individuals: HG00173, HG00174, HG00176, HG00177, HG00178, HG00179, and HG00180. Among these, FIN1 contains only HG00173, FIN2 contains two individuals, HG00174 and HG00176, and FIN4 contains the remaining 4 individuals HG00177, HG00178, HG00179, and HG00180. Finnish individuals were selected from the 1000 Genomes Project due to the availability of long and uniform length paired-end read sets with good coverage. For the experiments, we have only used reads from the 7 Finnish individuals, which are paired-end and 108 base pairs on each end. The depth of coverage of FIN1 is 4.25×, FIN2 is 8.04× and FIN4 is 15.87×. The hg19 human reference genome multi-fasta dataset is used as the reference with the default reference indexing scheme used in all comparisons between mapping tools. The hg19 reference genome dataset we used contained contigs for all autosomes and sex chromosomes, chromosome M as well as 29 alternative contigs for various chromosomes and 39 additional “chrUn” haplotypes that were not placed in any reference chromosome.

Chromosome 20 read datasets used in the experiments in Subsection 2.3.2 are from 32 Finnish individuals in the 1000 Genomes Project, with their reads restricted
to chromosome 20. The individuals included are: HG00171, HG00173, HG00174, HG00176, HG00177, HG00178, HG00179, HG00180, HG00182, HG00183, HG00185, HG00186, HG00187, HG00188, HG00189, HG00190, HG00266, HG00267, HG00269, HG00270, HG00272, HG00306, HG00311, HG00312, HG00357, HG00361, HG00366, HG00367, HG00369, HG00372, HG00373, and HG00377. The original read datasets all contained paired-end 108bp long reads. For the experiments with the paired-end read lengths of 70bp, 80bp, 90bp and 100bp, we trimmed each mate from the end that is closer to the fragment center. The total number of reads in these 32 datasets is \( \sim 37.27 \text{ million} \), with total depth coverage of 127.7\( x \), 118.3\( x \), 106.5\( x \), 94.6\( x \), 82.8\( x \) for the 2\( \times \)108bp, 2\( \times \)100bp, 2\( \times \)90bp, 2\( \times \)80bp and 2\( \times \)70bp datasets, respectively.

Simulated NGS datasets used

For the simulation results shown in Figures 2-1(c), 2-5, and 2-6, we used the human reference genome (hg19) with a fixed mutation rate of 0.1%, in order to capture the \( k \)-mer redundancy profile of the diploid human genome in the presence of mutations. Using this mutated reference, we simulated 20 million paired-end reads (2\( \times \)100bp) from chromosome 20 with varying sequencing error rates: 2\%, 1\%, 0.5\%, 0.25\%, and 0.125\%. We used SAMtools wgsim tool [62] for simulating reads.

Mouse datasets

In addition to human read mapping experiments, we also present results on the mouse genome. For mouse experiments, we used the original mm9 reference genome (NCBI build 37) as downloaded from UCSC genome bioinformatics site. Our mouse read datasets are taken from the Mouse Genomes Project [63]: ERR118246, ERR118251, ERR118256, and ERR118261, which each consist of paired-end 100bp Illumina HiSeq 2000 reads. The accession code for the datasets is ERA123494. The merged dataset, including all four datasets, contains \( \sim 122 \text{ million} \) reads with \( \sim 9 \times \) depth coverage.
2.3.2 Benchmarking experiments using real human data

We implemented compressively-accelerated versions of BWA aln and Bowtie2 (denoted as CORA-BWA and CORA-Bowtie2). We chose BWA and Bowtie2 for our experiments because they are widely used and also the primary means by which many biotechnology labs map large NGS read datasets. We also perform runtime and sensitivity comparisons to other mapping tools, including mrsFAST-Ultra, BWA mem, GEM, and Masai. All-mapping experiments are run on a single CPU of a 12-CPU Intel Xeon X5690 machine with 94GB RAM.

To determine whether compression yields acceleration, we compared the read alignment performance of our CORA-based compressively accelerated mappers against other methods using 1000 Genomes sequencing datasets from multiple individuals. These datasets consist of paired-end reads (108 bp) from the 1000 Genomes Phase 1 Illumina sequencing read data (see 2.3.1 Datasets used). Three of these datasets contain respectively 1, 2, and 4 different Finnish individuals with roughly 4×, 8× and 16× depth-coverage. We mapped these three read datasets onto the whole human reference genome (hg19) with four different alignment strategies: all-mapping with indels (Levenshtein distance), all-mapping without indels (Hamming distance), best-mapping with indels, and best-mapping without indels. We tested eight read mapping methods (BWA, CORA-BWA, Bowtie2, and CORA-Bowtie2, BWA mem, mrsFAST-Ultra, GEM and Masai) and measured their runtime and sensitivity. While we excluded Masai and GEM from the all-mapping experiments in Figure 2-1 due to their very low sensitivity (~10%) as compared to other tools, we do however provide a detailed analysis of their all-mapping performance in 2.3.5 Additional experimental results.

Whereas all CORA runtimes are reported on the full datasets, some of the other runtime and sensitivity results were estimated using downsampled read sets (see 2.3.6 Details of experimental setup).
All-mapping performance comparison with existing methods.

CORA’s compressive mapping framework achieved from ~6 times to 3 orders of magnitude speed-up compared to existing all-mappers with comparable sensitivity (Figure 2-1).

For the gapped alignment of the FIN1 dataset with roughly 4x coverage, we estimated that BWA aln would take more than 21 days to find paired-end all-mappings within a Levenshtein distance of 4 for each end, whereas CORA-BWA took less than 10 hours, more than 54 times faster than the original BWA mapper. For the FIN4 dataset, CORA-BWA was more than 62 times faster than BWA aln, indicating that CORA’s compressive acceleration increases with higher read-depth coverage. For the ungapped mapping of FIN4, CORA-BWA was still faster by an order of magnitude compared to BWA. CORA-BWA achieved these massive accelerations of BWA while also substantially improving the sensitivity of gapped and ungapped all-mapping (Figure 2-1(b)).

For the gapped alignment of the FIN1 dataset, we estimated that finding all-mappings using Bowtie2 would require several years to complete, whereas our compressively accelerated version of Bowtie2 was able to complete within 10 hours, effectively boosting Bowtie2’s all-mapping efficiency by three orders of magnitude, while again improving sensitivity.

For ungapped all-mapping on the FIN4 dataset CORA-BWA achieved ~6 times the speed up as compared to mrsFAST-Ultra, with near-perfect sensitivity. CORA-BWA reported mappings ~3.5 times faster even when mrsFAST-Ultra printed read mappings in an unordered fashion, creating computational debt for the downstream analysis. Furthermore, CORA’s acceleration as compared to mrsFAST-Ultra is significantly greater for higher redundancy reference genomes (e.g. mouse), effectively producing all-mapping results about an order of magnitude faster (see 2.3.4 Experiments on mouse data). CORA’s superior all-mapping speed as compared to mrsFAST-Ultra is remarkable, given that mrsFAST-Ultra is cache-optimized for all-mapping and the current implementation of CORA-framework does not employ any machine architec-
Sensitivity results for BWA, Bowtie2, mrsFAST-Ultra, CORA-BWA, and CORA-Bowtie2 are given in Figure 2-1(b) for the all-mapping alignment strategy. For ungapped all-mapping, sensitivities are computed with respect to the complete all-mapping datasets accepted as ground-truth, generated by exhaustive search. CORA-based versions of BWA and Bowtie2 achieved almost perfect sensitivity, whereas the original mappers exhibited a significant loss in sensitivity. For gapped all-mapping, as no method could be used as a perfect sensitivity ground truth, we compared the number of gapped mapping results different mappers reported. Both CORA-BWA and CORA-Bowtie2 have significantly higher sensitivity than BWA and Bowtie2's gapped all-mappers. This improvement is due to CORA's high-resolution homology table, particularly its representation of all reference-related homologies relevant to read-mapping, as well as its recovery scheme for missing 'read links'.

While we did not include Masai and GEM mappers in our main all-mapping comparisons due to low sensitivity, we performed benchmarks upon them as well. See 2.3.5 Additional experimental results for a detailed comparative analysis of GEM and Masai's gapped and ungapped all-mapping performance with CORA-BWA's.

Best-mapping performance comparison with existing methods.

While relatively modest compared to its acceleration of all-mapping, CORA still achieved substantial best-mapping performance improvements compared to existing state-of-the-art best-mappers for both gapped and ungapped mapping in terms of speed, sensitivity or both (Figure 2-9).

Remarkably even CORA's all-mappers (Figure 2-1(a)) reported mapping results faster than BWA, Bowtie2 and mrsFAST-Ultra's best-mappers. This is a considerable strength of the CORA framework, as it enables NGS analysis pipelines based on all-mapping that are even faster than existing pipelines based on best-mapping.
Figure 2-9: Runtime and sensitivity comparison results for whole genome ungapped (substitution-only) and gapped (with indels) best-mapping of 1000 Genomes Phase 1 Illumina 2x108bp paired-end read datasets of 4 Finnish individuals with ~16x read depth-coverage; similarity threshold is defined as Hamming distance of 4 for each end for the ungapped mapping and Levenshtein (edit) distance of 4 for each end for the gapped mapping benchmarks. Paired-end insert size interval is defined to be between 150 and 650 base pairs. We compared best-mapping runtimes of Bowtie2, BWA aln, BWA mem, mrsFAST-ultra (only for ungapped), GEM (only for gapped) and Masai against compressively accelerated version of BWA with two different modes: CORA-BWA and CORA-BWA-fast, which sacrifices some sensitivity allowing for faster best-mapping. The bars in the upper panel represent mappers' runtime performance, whereas the bars in the lower two panels indicate sensitivity performance: percentage sensitivity for ungapped mapping and number of mappings for gapped mapping. As BWA mem does not accept insert size intervals, we show two different mapping sensitivity measurements for it: sensitivity within the defined insert size interval and the increased sensitivity when mappings outside of the interval are included as well (the difference is indicated by the hatch pattern). Some of the results are estimated from a down-sampled set of reads; detailed benchmark criteria as well as mapping parameters can be found in Further details on experimental setup. The plots indicate that compared to the fastest best-mappers we tested against, CORA-BWA mappers are at least ~2x faster with superior or comparable sensitivity. The only mapper that approached CORA-BWA in terms of best-mapping runtime was Masai for the ungapped mapping experiment, albeit with drastically lower sensitivity; even then CORA-BWA-fast was >1.4x faster than Masai using less memory.
Figure 2-9: (cont.) Moreover, compared to the original BWA aln, CORA-BWA generated best-mapping results with near-identical sensitivity, but >3.2× faster for gapped and >3.1x faster for ungapped mapping. Furthermore, comparisons with CORA’s all-mapping runtime results in Figure 2-1(a) reveal that CORA can perform near-perfect sensitivity all-mapping faster than BWA, Bowtie2 and mrsFAST-Ultra can report best-mapping results. The peak memory usage of Bowtie2 was 3.2GB, BWA used 4.7GB and 6.2GB respectively for aln and mem, mrsFAST-Ultra used 4.7GB, whereas GEM and Masai’s memory usages were 4.1GB and 23.2 GB respectively. CORA-BWA, at the maximum of collapsing, coarse-mapping, and homology table traversal stages, used 19.7GB of memory for the runs that only utilized the exact homology table, whereas it used 64.1GB for the runs that also loaded the inexact homology table into memory (e.g. ungapped mapping with CORA-BWA default mode).

Table 2.4: All-mapping and coarse mapping runtimes of CORA-BWA when mapping paired-end 70-108bp read-length NGS datasets onto hg19 chromosome 20 within a Hamming distance of 2, 4, and 6.

<table>
<thead>
<tr>
<th>Hamming</th>
<th>2x70bp</th>
<th>2x80bp</th>
<th>2x90bp</th>
<th>2x100bp</th>
<th>2x108bp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full</td>
<td>779.02</td>
<td>843.40</td>
<td>905.61</td>
<td>954.09</td>
</tr>
<tr>
<td></td>
<td>Coarse</td>
<td>237.10</td>
<td>314.26</td>
<td>369.37</td>
<td>462.74</td>
</tr>
<tr>
<td>Hamming 2</td>
<td>1977.11</td>
<td>1863.28</td>
<td>1775.27</td>
<td>1920.49</td>
<td>1895.81</td>
</tr>
<tr>
<td></td>
<td>Coarse</td>
<td>994.67</td>
<td>1065.52</td>
<td>1095.38</td>
<td>1298.04</td>
</tr>
<tr>
<td>Hamming 4</td>
<td>3698.55</td>
<td>3395.73</td>
<td>3391.54</td>
<td>3541.02</td>
<td>3471.02</td>
</tr>
<tr>
<td></td>
<td>Coarse</td>
<td>1306.36</td>
<td>1795.29</td>
<td>2305.34</td>
<td>2597.97</td>
</tr>
</tbody>
</table>

Runtime and sensitivity analysis for varied k-mer length and error rate.

We performed additional experiments on real NGS data in order to demonstrate mapping runtime and sensitivity performance of CORA for varied k-mer lengths and error rates.

We aligned chromosome 20 reads for 32 Finnish individuals from 1000 Genomes Project onto hg19 chromosome 20 with a Hamming distance of 2, 4, and 6. The paired-end read lengths used were 108bp, 100bp, 90bp, 80bp and 70bp. For these error rates and read lengths, we constructed homology tables with the corresponding k-mer lengths of 54bp, 50bp, 45bp, 40bp and 35bp, with a Hamming distance threshold of 1, 2, and 3, respectively. Table 2.4 shows the runtime of these experiments in seconds.

We also estimated the mapping sensitivity for a subset of the experiments above, for the paired-end 90bp, 100bp and 108bp read datasets across Hamming distance thresholds of 2, 4 and 6 (Table 2.5).

For the experiments in which the homology table distance thresholds were twice
Table 2.5: All-mapping percentage sensitivity results of CORA-BWA when mapping paired-end 90-108bp read-length read datasets onto hg19 chromosome 20 within a Hamming distance of 2, 4, and 6. For mapping experiments with Hamming distance 4 and 6, we also measured the percentage sensitivity of CORA when recovering mappings with a Hamming distance of 2. For experiments with Hamming distance 6, we also looked at the sensitivity of mapping with Hamming distance 4.

<table>
<thead>
<tr>
<th>Homology table distance threshold</th>
<th>Search distance threshold</th>
<th>2 x 90bp</th>
<th>2 x 100bp</th>
<th>2 x 108bp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming 2</td>
<td>Hamming 2</td>
<td>99.523%</td>
<td>99.433%</td>
<td>99.289%</td>
</tr>
<tr>
<td>Hamming 4</td>
<td>Hamming 2</td>
<td>99.996%</td>
<td>99.996%</td>
<td>99.997%</td>
</tr>
<tr>
<td></td>
<td>Hamming 4</td>
<td>99.506%</td>
<td>99.380%</td>
<td>99.218%</td>
</tr>
<tr>
<td>Hamming 6</td>
<td>Hamming 2</td>
<td>99.999%</td>
<td>99.999%</td>
<td>99.999%</td>
</tr>
<tr>
<td></td>
<td>Hamming 4</td>
<td>99.910%</td>
<td>99.857%</td>
<td>99.791%</td>
</tr>
<tr>
<td></td>
<td>Hamming 6</td>
<td>99.421%</td>
<td>99.250%</td>
<td>99.024%</td>
</tr>
</tbody>
</table>

Table 2.6: Runtime and sensitivity results of all-mapping with CORA-BWA when mapping a paired-end 2x108bp NGS read dataset onto hg19 chromosome 20 when recovery scheme is disabled for Hamming distance thresholds of 2, 4 and 6.

<table>
<thead>
<tr>
<th>Homology table distance threshold</th>
<th>Performance measurement (no mapping recovery mode)</th>
<th>2 x 108bp performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamming 2</td>
<td>Hamming 2 search sensitivity</td>
<td>96.323%</td>
</tr>
<tr>
<td></td>
<td>Full runtime (seconds)</td>
<td>1082.37</td>
</tr>
<tr>
<td>Hamming 4</td>
<td>Hamming 2 search sensitivity</td>
<td>99.930%</td>
</tr>
<tr>
<td></td>
<td>Hamming 4 search sensitivity</td>
<td>89.485%</td>
</tr>
<tr>
<td></td>
<td>Full runtime (seconds)</td>
<td>1854.053</td>
</tr>
<tr>
<td>Hamming 6</td>
<td>Hamming 2 search sensitivity</td>
<td>99.982%</td>
</tr>
<tr>
<td></td>
<td>Hamming 4 search sensitivity</td>
<td>97.922%</td>
</tr>
<tr>
<td></td>
<td>Hamming 6 search sensitivity</td>
<td>84.123%</td>
</tr>
<tr>
<td></td>
<td>Full runtime (seconds)</td>
<td>3410.816</td>
</tr>
</tbody>
</table>

or more than the search distance threshold, CORA performed mapping with almost 100% sensitivity. This observation is in accordance with the mathematical proof given earlier. In practice, a small sensitivity loss of <0.004% is observed due to potential misses in the coarse mapping stage using BWA aln.

Recovery scheme analysis for varied k-mer length and error rate.

We also ran CORA without the recovery scheme for the original 108bp paired-end read dataset in order to analyze the tradeoff between runtime and sensitivity in comparison to the runs above with the recovery scheme enabled (Table 2.6).
Performance results indicate that the recovery scheme becomes more important as the homology table distance threshold increases, especially if the search distance threshold is larger than half the homology table distance threshold. The runtime was nominally improved when the recovery scheme was disabled.

2.3.3 Experiments on simulated human data

For the runtime results that demonstrate CORA’s enhanced performance with improvements in quality of sequencing (Figure 2-1(c)), we simulated 20 million paired-end reads (2x100bp) from chromosome 20 of the human reference genome (hg19) with a fixed mutation rate of 0.1% and varying sequencing error rates of 2%, 1%, 0.5%, 0.25%, and 0.125%. We used SAMtools wgsim tool for simulating paired-end reads.

For sequencing read simulation, we employed SAMtools version 0.1.19 with default insert size and distribution parameters ‘-d 500 -s 50’, as well as ‘-R 0 -X 0’ parameters in order to simulate read errors within the Hamming distance metric.

To demonstrate CORA’s sublinear coarse mapping scaling property, we further simulated four datasets with one billion paired-end reads from chromosome 20 with 2%, 0.5%, 0.125%, and 0% sequencing error rates (denoted as E2, E05, E0125, and E0 datasets).

Even though any read-mapping method must scale at least linearly with the number of lines in the input dataset as well as the output, between the input reading and output writing stages, the CORA framework achieves sublinear scaling for costly sequence similarity computations. In fact, in the absence of sequencing errors, the total number of read sequences that are processed in the coarse mapping stage is in effect constant above 500× read depth-coverage (Figure 2-5). Notably, the rate of redundancy in read k-mers monotonically increases even in the presence of sequencing errors (Figure 2-6).
2.3.4 Experiments on mouse data

We benchmarked CORA-BWA in comparison to mrsFAST-Ultra and BWA aln/mem for substitution-only all-mapping and best-mapping for mouse read datasets using a single CPU of a 20-CPU Intel Xeon E5-2650 (2.3GHz) machine with 384GB 2133MHz RAM. For all-mapping, we ran CORA-BWA on the full read dataset, whereas we ran mrsFAST-Ultra on a 1/10 downsampled dataset due to prohibitive runtime. All-mapping sensitivity comparisons were measured on the downsampled dataset, based on the assumption that mrsFAST-Ultra produces mappings with perfect sensitivity. For best-mapping, we ran CORA-BWA on the full dataset with both fast best-mapping mode, which uses only the exact homology table, and original best-mapping mode, which makes use of both exact and inexact homology, whereas we ran BWA aln and BWA mem on the 1/10 subsampled dataset.

The mouse read dataset consisted of 4 individuals with 100bp paired-end read datasets with a total of 8.96× depth coverage. We mapped these read datasets onto the mm9 mouse reference genome within a Hamming distance of 4 for each end and with an allowed insert size interval of 150-650bp. CORA’s homology table of mm9 reference genome was constructed with a k-mer length of 50bp and a Hamming distance threshold of 2. As the homology table is precomputed, its runtime requirement is not included in our comparisons.

When scaled to the whole read dataset from the 1/10 downsampled runs, mrsFAST-Ultra’s runtime corresponds to 63.4 hours for mapping and 72.2 hours for sorting the dataset; 135.6 hours for the full mapping pipeline for the all-mapping reads to be used for downstream analysis. Scaling for mapping and sorting components were performed separately as described in 2.3.6 Details of experimental setup. In terms of sensitivity, we assumed mrsFAST-Ultra’s ungapped all-mapping sensitivity to be 100%. In comparison, CORA-BWA performed all-mapping on the full dataset in 14.8 hours (readily in read sorted order), capturing 99.64% of the mappings mrsFAST-Ultra produced. This result indicates that, CORA can produce read sorted all-mapping results ~9.2× faster than the full mrsFAST-Ultra pipeline with near-perfect sensitivity.
(~4.3× faster if sorting is not included).

For the 1/10 downsampling run, BWA aln completed reporting substitution-only best-mapping results in 181 minutes and 55 seconds, corresponding to 30.3 hours when scaled to the full run. In comparison, CORA completed performing best-mapping on the full dataset in 12.7 hours for the original best-mapping mode and 6.78 hours for its fast best-mapping mode. BWA aln reported best-mapping results with ~95.02% sensitivity in the downsampling read dataset, whereas CORA-BWA's original best-mapping mode reported mappings with 99.41% sensitivity for the same set of reads, ~2.4× faster than BWA aln. CORA’s fast best-mapping mode, which does not make use of the inexact homology table, reported mapping results with 92.5% sensitivity, ~4.47× faster than BWA aln.

2.3.5 Additional experimental results

All-mapping benchmarks with GEM and Masai

We provide detailed results here for both Masai and GEM's all-mapping performance and compare them to CORA's all-mapping results.

For our benchmarking experiments with Masai, we used version 0.7.1 for Linux x86_64. As precomputation, we indexed the hg19 reference genome using Masai's default indexing algorithm. We used the command line arguments '-mm all' for all-mapping, '-e 4' for mapping with distance 4 for each end, and '-ll 400 -le 250' which determines the valid paired-end insert size interval as [150, 650]. For the ungapped (substitution-only) experiments, we declared the flag '-ng'; for gapped mapping experiments we did not declare this flag. The remainder of the parameters were chosen by default.

Similar to BWA, Masai requires three separate executions for paired-end mapping: single-end mapping for each end and a third execution for inferring paired-end mappings from the two intermediate files. Masai crashed during each of these 3 stages on the full FIN4 dataset, for both substitution-only and gapped mapping runs, throwing 'std::bad_alloc' errors. While the last stage crashed after saturating the entire 96GB
of memory, the first two stages crashed after using \( \sim 60 \) GB of memory.

For this reason, we uniformly downsampled the FIN4 dataset in order to gauge Masai’s all-mapping speed and sensitivity. When we ran it on a 1/100 downsampled dataset for ungapped (substitution-only) all-mapping, Masai was able to successfully report mapping results in 33 minutes and 51 seconds, corresponding to 56.4 hours for the full run. The percentage sensitivity of Masai for the downsampled set was 10.1\%. In comparison, CORA-BWA takes 14.4 hours to run on the full dataset while producing mappings with 99.7\% sensitivity. For ungapped mapping, Masai produces \( \sim 10\% \) of the ungapped mappings that CORA-BWA reports while taking 3.9x times the time CORA takes to report the mappings.

For gapped all-mapping on the 1/100 downsampled dataset, Masai produced mapping results in 67 minutes and 27 seconds, which corresponds to 114 hours for the full run. Strikingly, Masai’s sensitivity at detecting alignments was substantially lower than other all-mappers we benchmarked in that it produced 4.57 million valid mappings for the 1/100 dataset (with the criteria of valid gapped mappings defined earlier in Subsection 2.3.2). In comparison, CORA-BWA takes 30.95 hours for mapping the full FIN4 dataset and produces 45 million valid mappings for the same set of reads included in the subsampled Masai run. Overall, Masai produces 10.1\% of the mappings CORA-BWA produces while taking \( \sim 3.68 \times \) the time to compute the mappings.

For our benchmarking experiments with GEM, we used GEM-mapper build 1.376. As precomputation, we indexed the hg19 reference genome using GEM’s default indexing algorithm. We consecutively ran gem-mapping and gem-2-sam algorithm to create a pipeline that takes in FASTQ read input and reports SAM mapping output. We used command line arguments ‘-E 4’ for setting the edit distance threshold to 4 for each end, ‘-b’ to align both ends, and ‘-s 4’ for the mapper to recover all possible mapping edit distance strata that cover 4 edits. We also defined ‘--min-insert-size 150’ and ‘--max-insert-size 650’. Our full FIN4 data runs as well as subsampled runs consistently crashed using the GEM mapper, reporting a ‘wrong alignment’ crash error after processing \( \sim 1\% \) of the reads. Fortunately, the GEM mapper reports mapping results for each read as it computes mapping loci, so it is possible to give a rough
estimate of its sensitivity and runtime on the full dataset. Our estimated runtime of
the GEM mapper on the full FIN4 dataset is 81.7913 hours. The number of valid
gapped mappings GEM produces for the processed reads is 3.385 million. In com-
parison, CORA-BWA produces 37.185 million valid gapped mappings for the same
dataset. Overall, GEM produces 9.1% of the mappings that CORA-BWA reports
while taking ~2.64× more time.

Note that for seed-and-extend read mapping algorithms, improving sensitivity
affects runtime superlinearly [64], as increasing sensitivity requires sampling shorter
seeds from the read with drastically higher chance occurrences. Thus, we can suppose
that a more sensitive Masai algorithm, using similar data structures but performing a
deeper index search with shorter seeds, would be at least ~ 36× slower than CORA-
BWA for a similar level of sensitivity; a more sensitive GEM algorithm would be at
least ~ 29× slower for gapped all-mapping.

Oculus benchmarks

We tested Oculus, a mapping acceleration tool that wraps off-the-shelf aligners and
leverages compression of fully identical reads, on the FIN1 dataset with BWA aln.
The total runtime was 31.4 hours with 16 minutes spent within Oculus. The estimated
saved time reported by Oculus was 37.8 minutes, which corresponds to a <2% runtime
improvement. This limited acceleration is due to the fact that Oculus requires both
ends of a paired-end read to be identical, which is rare even in large datasets.

2.3.6 Details of experimental setup

We compared mapping speed and sensitivity of BWA, Bowtie2, compressively accel-
erated BWA (CORA-BWA) and compressively accelerated Bowtie2 (CORA-Bowtie2)
for the all-mapping and best-mapping alignment strategies. For our comparisons, we
used Bowtie2 version 2.1.0, BWA version 0.7.5a, mrsFAST-Ultra version 3.3, GEM
version 1.376, Masai version 0.7.1 and Oculus version 0.1.2.

Mapping criteria and evaluation of sensitivity. For mapping benchmarks pre-
sented, we used the Hamming distance (substitution distance) threshold of 4 for ungapped and Levenshtein distance (edit distance) threshold of 4 for gapped. We required both mates of the paired-end reads to be aligned end-to-end with an insert size interval between 150 and 650 base pairs, which specifies the distance from the beginning position of the first mate in the reference to the second’s (i.e. the alignment gap length between mates + single read length).

We specified this alignment criteria for both best-mapping and all-mapping, so that we can compare CORA’s all-mapping performance not only with other all-mapping methods but also the best-mapping performance of existing tools.

For the all-mapping benchmarks, we measured each mapper’s ability to report all of the mappings that satisfy the criteria given above. For example, any paired-end read alignment with an insert size between 150bp and 650bp and with 4 substitutions for each end is a valid alignment to be reported for ungapped all-mapping (or \(x\) insertions, \(y\) deletions and \(z\) substitutions for gapped all-mapping, such that \(x+y+z = 4\)).

Note that in our gapped mapping experiments the penalty for each mismatch and each base of a deletion/insertion is chosen to be equal, as specified by the Levenshtein distance metric, which is both the most basic gapped distance metric and the only one uniformly supported by the mappers we tested. In the cases where the default gapped mapping mode of a mapper did not exactly correspond to Levenshtein distance metric, we specified a gap opening penalty of 0 and a gap extension penalty equal to the mismatch penalty for the mappers to ensure a consistent comparison across different methods. While this is a fair comparison benchmark that evaluates a desired property of mappers, the ability to sensitively detect high-quality mappings of reads within a specified distance threshold, it does not evaluate different mappers’ ability to detect mappings outside of the specified distance threshold (mappings containing more mismatches or covering indels longer than the threshold value). In order to have an estimate of what percentage of variants this would exclude from our analysis, we recomputed the NA12878 genotyping sensitivities presented in Yu et al. [65] for both GATK and SAMtools pipelines (GATK ‘best-practices’ bundle used as gold-
When genotyping performance of BWA's default mapping results was compared to the subset of these mappings within Levenshtein distance of 4, our gapped mapping benchmark criteria, we see that 2.7% of the SNPs as well as 7.5% of the indels in the gold-standard change from true-positive calls in the former to false-negative calls in the latter. However, the decrease in sensitivity caused by excluding mappings outside of Levenshtein distance of 4 also result in a substantial increase in precision: False discovery rate within the top 2.5 million SNP calls is 24.6% higher in the full mapping dataset compared to the filtered. In the case of top 150 thousand indels, this increase in FDR is 16.1%. In other words, by selecting a Levenshtein distance threshold of 4, our benchmarks restrict the solution space to a smaller yet higher quality set of mappings that result in higher precision but lower recall variant calls.

Sensitivity for ungapped all-mapping results is calculated as the percentage of these valid paired-end read alignments that are correctly reported by the mapping method. Note that sensitivity for all-mapping is equivalent to the accuracy, as read mapping tools in general (including CORA-accelerated versions) do not falsely report a mapping within the specified similarity threshold. For ungapped all-mapping sensitivity benchmarks, mapping results of mrsFAST-Ultra is accepted as ground truth as it can perform ungapped mapping with perfect sensitivity.

Sensitivity for gapped all-mapping results is not calculated as percentage results as none of the mappers we tested were able to give all-mapping results with perfect sensitivity for a gapped alignment strategy. For that reason, we compared the number of valid non-redundant mappings that each gapped mapper was able to map within the specified insert length interval and error distance. We define a mapping to be redundant if there is another mapping for the same read at the same loci for both mates with a potentially different traversal of the dynamic programming matrix for either end. For example, if there is a read mapping with no errors at position $x$, any other mapping for the same read at position $x$ is considered redundant. Moreover, mappings for the same read that start at position $x + k$ with $k$ inserted bases (or $k$ silenced bases) at the beginning of the read are also considered redundant. Fur-
thermore, if an insertion or deletion sequence spans a micro-repeat region in the read or the reference, indels spanning different instances of the repeat are all considered redundant apart from one of them with a minimal error distance. Since there can be many non-trivial configurations of these redundant mapping variants as well as their combinations, we employed a simpler scheme to obtain a non-redundant all-mapping set.

1. Each reported paired-end mapping was merged into a single line in the SAM output.

2. Each reported paired-end mapping was merged into a single line in the SAM output.

3. Starting from the second line of the sorted file, if the previous line's read name and chromosome are identical with the current line, the positions of the first mates are at most $d$ apart between the two lines, and the positions of the second mates are also at most $d$ apart ($d$ being the Levenshtein distance threshold of mapping), the mapping in the current line is considered redundant and eliminated.

Though this simple elimination scheme does not necessarily pick the lowest distance mapping, it is adequate for measuring the total number of non-redundant all-mappings. In very few cases, this elimination scheme can cause false elimination of non-redundant mappings. However, we estimated this effect to be negligible (<0.001%).

The ungapped best-mapping read alignment scheme is defined as reporting only one of the valid paired-end mappings with the lowest cumulative Hamming distance with respect to both ends. In the case of multiple valid paired-end mappings with the lowest Hamming distance, any one of the mappings with the lowest distance can be selected as a valid best-mapping output. Sensitivity for ungapped best-mapping results is calculated as the percentage of reads that have at least one valid mapping to the reference, and one of the lowest Hamming distance mappings is reported by the
alignment method. For ungapped best-mapping sensitivity benchmarks, the lowest Hamming distance tier of mrsFAST-Ultra’s mapping results is accepted as ground truth as it can perform ungapped mapping with perfect sensitivity.

Sensitivity for gapped best-mapping results is not calculated as percentage results as none of the mappers we tested were able to give best-mapping results with perfect sensitivity for a gapped alignment strategy. For this reason, sensitivity results are reported as the total number of reads that the mappers reported a valid paired-end mapping for within the specified Levenshtein distance threshold and insert size interval.

In the specific case of BWA mem, as it does not allow the user to specify an insert size interval, we measured the sensitivity for two sets of criteria: (1) using the default criteria described above and (2) including additionally mappings BWA mem reported outside of the insert size interval so long as other tools also found a mapping for the same read within the insert size interval. The reason for this two-tiered sensitivity analysis is that, while our original benchmark provides a head-to-head comparison of all tools according to the same mapping criteria it also indirectly penalizes BWA mem’s lack of a user option for insert size intervals. As a way to ameliorate this situation, our second sensitivity analysis aims to be more generous to BWA mem by incorporating its mappings outside of the insert size interval to the analysis as well, while still being fair to other mappers by not incorporating mappings where there are no mappings for the same read within the paired-end insert size interval (see 2.3.6 Experimental setup for best-mapping benchmarks for details).

Experimental setup for all-mapping benchmarks.

The all-mapping scheme for BWA is employed as follows:

0. Reference genome is indexed using default parameters of BWA.

1. First mate of the paired-end read is aligned with the ‘bwa aln’ program using parameters ‘-n 4’ (specifying the maximum Hamming/Levenshtein distance for each valid read alignment as 4), and ‘-N’ for identification of all possible valid
alignments for each mate. For ungapped mapping, `-o 0' was specified for preventing gap openings and indels. For gapped mapping `-d 0 -i 0' was defined in order to sensitively detect all indel mappings for the entire read.

2. Second mate is aligned using the same command and parameters.

3. Paired-end alignments are found with the ‘bwa sampe’ program using parameters `-s' for disabling Smith-Waterman for the unmapped mate in order to restrict the reported mappings to a pre-defined Hamming or Levenshtein distance threshold for both mates, `-c 0' for eliminating chimeric read mappings, `-a 650' for reporting paired-end read alignments with at most 650 base pairs insert size (BWA sampe does not support a lower bound for insert size), and `-n 1M -N 1M -o 100M' for enabling maximum possible sensitivity of read mappings (M stands for one million).

In addition to these three stages, as BWA sampe reports only a single paired-end mate in its output and the remaining read mappings are reported as single-end mappings without any pair information, we implemented a paired-end read mapping extraction tool for BWA that efficiently parses the locations reported by sampe in XA field for each end, sorts each list of single-end mappings and performs a linear scan to report all pairs of locations with matching chromosomes and with mates within the user-specified insert size interval. In our reported runtimes, we indicated the cost of this paired-end mapping extraction tool with the lighter color on top of the BWA bars. We did not include its costs for indexing the genome. For downsampled runs, we extrapolated the full runtime of BWA assuming linear scaling with the read dataset size.

The all-mapping scheme for Bowtie2 is employed as follows:

0. Reference genome is indexed using default parameters of Bowtie2.

1. Bowtie2 aligner is run to perform paired-end alignment of the read dataset onto the reference genome using parameters ‘-no-mixed’ in order to suppress alignments that are not paired-end, `-a -end-to-end -ignore-quals' in order to
find end-to-end mappings of each mate using the Hamming or Levenshtein distance metric and report all paired-end mappings, ‘-np 6 -mp 6 -score-min L,0,-0.25’ for limiting the maximum number of allowed number of substitutions/insertions/deletions to only 4 bases for each 108bp long mate, and ‘-minins 150 -maxins 650’ specifying the insert size interval. For ungapped mapping, ‘-rdg 1000,1000 -rfg 1000,1000’ was defined in order to eliminate all indels. For gapped mapping ‘-rdg 0.6 -rfg 0.6’ was defined to set equal weight between indels and substitutions corresponding to Levenshtein distance.

In our reported runtimes for Bowtie2, we have not included its cost for indexing the genome. For downsampled runs, we extrapolated the full runtime assuming linear scaling with the read dataset size.

The ungapped all-mapping scheme for mrsFAST-Ultra is employed as follows:

0. Reference genome is indexed using default parameters of mrsFAST-Ultra.

1. Parameters ‘-e 4 -min 150 -max 650’ are specified for the insert length interval for the paired-end mapping as well as Hamming distance of 4 for each read end.

mrsFAST-Ultra reports all-mappings in an unsorted order (w.r.t. read or chromosome order) whereas all other tools we benchmarked, including CORA, report read-ordered mappings. Printing all-mapping results in an unsorted order creates computational debt for the downstream analysis tools, as they have to directly or indirectly perform a sorting task in order to identify all of the mappings of each read. We sorted mrsFAST-Ultra’s output by read names using linux sort. In order to perform efficient sorting we used the local temporary directory and declared LC_ALL=C environment variable, which dramatically improves performance for sorting files with ASCII characters. For downsampled runs, we extrapolated the mapping portion of mrsFAST-Ultra’s runtime linearly; the sorting portion of its runtime has complexity \( \Theta((N/M)\log R(N/M)) \) for the R-Way merge sort algorithm that linux sort implements, where N and M correspond to data and memory size, respectively, and R is the number of files the merge operation is performed on simultaneously. As
we did not measure the constants in the complexity function, including R, M or the constant factor before the complexity function, we estimated the cost of sorting for two separate file size as two independent data points for extrapolation. These two files were 1/10 downsamplend and 1/20 downsamplend mapping output files, both of which are larger than the RAM size of the machine. If $X$ is the cost of sorting the 1/20 downsamplend file and $Y$ is the cost of sorting the 1/10 downsamplend file, the cost of sorting the full file is estimated to take $20X + 10(Y - 2X)\log_2(20)$, which is independent from $R$, $M$ or the constant factor before the complexity function. In Figure 2-1 we indicated the cost of this sorting process with the lighter color on top of the mrsFAST-Ultra runtime bars.

Note that this sorting operation merely positions the all-mappings of each read to be adjacent in the file and does not preserve the original order of the read names or perform any sorting on the chromosome or the positions. Ideally a mapper should also preserve the order of the original reads as well as print the mappings of each read in proper order (with respect to chromosomes and positions); however, we assume resolving these will not incur significant computational costs for downstream analyses. Furthermore, we ignore any potential super-linear disk-operation costs that might arise from disk inefficiencies due to reading/writing larger files on disk. We also attempted to convert the sam file into bam format first and sort using SAMtools, but obtained a larger runtime cost even without extracting the sorted bam file.

For compressively-accelerated versions of BWA and Bowtie2, we generated the hg19 homology table using 54bp $k$-mers. We specified that compression is performed using 54bp $k$-mers (corresponding to half of each mate) and coarse mapping, using BWA aln and Bowtie2. We specified the homology table traversal stage to output all mappings within the Hamming distance limit of 4 for substitution-only mapping and the Levenshtein (edit) distance limit of 4 for gapped mapping. In reported runtimes, we have not included the preprocessing cost for homology table creation, nor the reference indexing costs incurred by BWA aln and Bowtie2. The runtime cost of $k$-mer based read compression is included in the total runtimes reported.
While all of the CORA all-mapping runtimes were measured from full dataset runs, some of the other tools were estimated using a downsampling read set. Downsampling was performed uniformly and consistently throughout the experiments: All 1/10 downsampling read sets are identical for the same dataset across different mapping experiments and the 1/10 downsampling read set is a superset of the 1/100 downsampling read set and so on.

Both Bowtie2’s gapped and ungapped runtimes were estimated using a 1/1000 downsampling read set due to prohibitive runtime cost. BWA’s gapped runtimes were estimated using 1/100 downsampling read sets. The number of mappings for the gapped all-mappers was estimated from the same 1/1000 downsampling read set for FIN4.

The runtime of mrsFAST-Ultra mapping and sorting was measured using a 1/10 downsampling read set. Sensitivity percentages and runtimes for BWA’s ungapped all-mapping were measured from a 1/10 downsampling read set, whereas Bowtie2’s ungapped all-mapping was measured from a 1/1000 downsampling read set. CORA mappers’ ungapped sensitivity percentages were measured from the full read set.

**Experimental setup for best-mapping benchmarks.**

The best-mapping scheme for BWA aln is employed as follows:

0. Reference genome is indexed using default parameters of BWA.

1. First mate of the paired-end read is aligned with BWA aln program using parameters ‘-n 4’ (specifying the maximum Hamming distance for each valid read alignment as 4). For ungapped mapping strategy ‘-o 0’ was used in order to prevent gap openings and indels. For gapped mapping we defined ‘-d 0 -i 0’.

2. Second mate is aligned using the same command line parameters as the first mate.

3. Paired-end alignments are found with BWA sampe program using parameters ‘-s’ for disabling Smith-Waterman for the unmapped mate in order to restrict the
reported mappings to pre-defined Hamming distance threshold for both mates, ‘-a 650’ for reporting paired-end read alignments with at most 650 base pairs insert size (sampe does not support lower bound for insert size), and ‘-n 0 -N 0’ in order to report only one paired-end mapping per read.

The best-mapping scheme for BWA mem is employed as follows:

0. Reference genome is indexed using default parameters of BWA.

1. BWA mem aligner is run with parameters ‘-A 1 -B 1 -O 0 -E 1 -L 1 -U 1000 -T 100’ for gapped mapping, which assigns a penalty of 1 for each edit operation, a score of 1 for each match operation, and a threshold score of 100 corresponding to Levenshtein distance of 4. For ungapped mapping, ‘-A 1 -B 1 -O 1000 -E 1000 -L 1000 -U 1000 -T 100’ is defined in order to prevent any insertions and deletions and set the distance threshold as Hamming distance of 4. For both alignment strategies, mate rescuing is disabled by specifying ‘-S’ which results in a major increase in speed with negligible loss in sensitivity, ‘-t 1’ is defined to run BWA mem on a single processor, and all other parameters were selected as default.

As BWA mem does not provide a user option for specifying paired-end insert size interval, we measured and reported two levels of sensitivity for it (Figure 2-9). BWA mem’s base sensitivity, which is lower, was measured using the default criteria employed to evaluate the other best mappers, whereas a second sensitivity measure, which is more tolerant for mappings outside the specified insert size interval, was computed by also taking into account BWA mem’s mappings outside of the [150, 650] insert size interval so long as the edit distance of the mapping reported, summation of both mates’ edit distances, was within the distance threshold of the original benchmarks and other tools also found a mapping for the same read within the [150, 650] insert size interval. In the case of ungapped mapping, we used the set of reads with valid mrsFAST-Ultra mappings for this comparison since mrsFAST-Ultra reports ungapped mappings with perfect sensitivity. In the case of gapped mapping,
the union of the set of reads with valid BWA aln, Bowtie2 or mrsFAST-Ultra mappings were used.

The best-mapping scheme for Bowtie2 is employed as follows:

0. Reference genome is indexed using default parameters of Bowtie2.

1. Bowtie2 aligner is run to perform paired-end alignment of the read dataset onto the reference genome using parameters ‘--no-mixed’ in order to suppress alignments that are not paired-end, ‘--end-to-end --ignore-quals’ in order to perform end-to-end mapping of each mate using Hamming or Levenshtein distance metric for mapping similarity, ‘--np 6 --mp 6 --score-min L,0,-0.25’ for limiting the maximum number of allowed number of substitutions to only 4 base-pairs for each 108bp long mate, ‘--minins 150 --maxins 650’ specifying the insert size interval. For ungapped mapping ‘--rdg 1000,1000 --rfg 1000,1000’ parameters were used. For gapped mapping, the same parameters were defined as ‘--rdg 0.6 --rfg 0.6’.

The ungapped best-mapping scheme for mrsFAST-Ultra was performed as follows:

0. Reference genome is indexed using default parameters of mrsFAST-Ultra.

1. Parameters ‘-e 4 -min 150 -max 650 -best’ are specified for best-mapping with the specified insert length interval for the paired-end mapping as well as Hamming distance of 4 for each read end.

As mrsFAST-Ultra prints best-mapping output in read sorted order, there is no sorting cost added for best-mapping.

The best-mapping scheme for Masai was performed as follows:

0. Reference genome is indexed using default parameters of Masai.

1. The FASTQ file containing the first mates are aligned with masai_mapper algorithm was call with ‘-e 4’ parameter that specifies Hamming/Levenshtein distance threshold. In the case of ungapped mapping we additionally specified parameter ‘-ng’ which prevents indels.
2. The FASTQ file containing the second mates is aligned using the same command line parameters as the first FASTQ file.

3. Paired-end alignments are found using masai_output_pe algorithm, with the arguments ‘-ll 400 -le 250’ which specify the insert size interval as [150, 650]. For both gapped and ungapped alignments, Masai crashed with a bad_alloc error on the full FIN4 dataset after using 144.2GB memory in stage 3. For this reason, we performed downsampled runs for Masai, extrapolating full runtime.

The gapped best-mapping scheme for GEM was performed as follows:

0. Reference genome is indexed using default parameters of GEM.

1. gem-mapper algorithm was called with the command-line arguments ‘-E 4 -b’ to map both ends within a Levenshtein distance of 4, ‘-T 1’ for restricting GEM to a single processor, ‘-min-insert-size 150 -max-insert-size 650’ in order to specify the insert size interval.

2. gem-2-sam algorithm was called in order to convert the mappings from GEM’s internal format to SAM format.

For best-mapping runs, the CORA-BWA framework is run with the same parameters as the all-mapping runs except for the last stage where the final mappings are inferred and only a single mapping with the lowest Hamming or Levenshtein distance is reported. For ungapped best-mapping, we performed runs with two separate speed levels: fast mode for which only the exact homology table is traversed; the default mode for which both exact and inexact homology tables are traversed. For gapped best-mapping, we also performed runs with two separate speed levels: fast sensitivity mode for which only the exact homology table is traversed and the banded dynamic-programming algorithm is not employed; default mode for which the banded dynamic-programming algorithm is also employed.

All best-mapping runtimes were measured from full dataset runs, apart from GEM and Masai which consistently crashed on the full dataset. We estimated Masai runtimes from a 1/10 uniformly downsampled read.
Similar to the all-mapping runs, best-mapping runs with GEM also crashed early in the read list, throwing a ‘wrong alignment’ error. We removed a portion of the reads that GEM could not process and reran GEM, but it crashed again soon after. For this reason, we extrapolated the full runtime and sensitivity of GEM from only the mapped portion of the dataset until the first crash, which corresponds to ~1% of the full read dataset.

All sensitivity percentages for ungapped best-mapping were estimated from the 1/10 downsampled FIN4 dataset. While all sensitivity results for gapped best-mapping were estimated from the 1/100 downsampled read set apart from GEM. We estimated GEM's sensitivity on the read set that it mapped and compared its sensitivity to CORA-BWA’s sensitivity on the same set of reads. We assumed this sensitivity ratio to be fixed when we estimated GEM’s sensitivity for the full dataset.
Chapter 3

Scalable Quality Score
Compression That Improves Accuracy

3.1 Overview

The bulk of NGS data typically consists of read datasets, in which each base call is associated with a corresponding quality score, which consumes at least as much storage space as the base calls themselves [3]. Quality scores are often essential for assessing sequence quality, filtering low quality reads, assembling genomic sequences, mapping reads to a reference sequence and performing accurate genotyping. Because quality scores require much space to store and transmit, they are a major bottleneck in any sequence analysis pipeline, impacting genomic medicine, environmental genomics, and the ability to find signatures of selection within large sets of closely related sequenced individuals. Early studies on compressing NGS datasets have mainly focused on compressing sequence data itself, aiming to leverage the inherent redundancy present in read sequences to reduce the space needed for storing ‘raw’ reads [66, 67, 68, 43, 69].

That said, the Phred quality scores encoding the “base-calling confidence” take
up more than twice the space on disk as the read sequence itself ($\sim 2.3x \text{ to } 2.8x$ for Illumina reads). Furthermore, it is more challenging to compress the quality scores [3], as they not only have a larger alphabet size, but also have limited repetitive patterns as well as little direct correlation with the bases sequenced.

Computational methods for NGS quality compression fall into two main camps: lossless compression methods that include general-purpose text-compressors, such as GZIP, BZIP2 or 7zip, as well as methods specialized to exploit the local similarity of quality values for further compression [70, 71, 69]; and lossy compression methods that aim to achieve further compression by sacrificing the ability to reconstruct the original quality values [72, 3, 73].

A recent area of investigation in quality score compression is exploiting sequence read information within NGS datasets in order to boost the compression of quality scores. Most of these methods need to compute expensive whole-genome alignments of the NGS read dataset, then use additional position and coverage information obtained from the alignments to compress quality values [43, 67, 74]. However, it has been shown that alignment-agnostic methods can also utilize sequence data to achieve better compression of quality values [75], but again require costly operations (e.g. BWT) to be run on the entire read dataset. Thus far neither approach has been able to truly address the scalability problem of quality score compression for terabyte-sized or larger NGS datasets.

This chapter describes how to recover quality information directly from sequence data using a novel compression method called Quartz, rendering such scores redundant and yielding substantially better space and time efficiencies for storage and analysis. Quartz is designed to operate on NGS reads in FASTQ format, but can be trivially modified to discard quality scores in other formats for which scores are paired with sequence information. Discarding 95% of quality scores counterintuitively resulted in improved SNP calling, implying that compression need not come at the expense of accuracy.

At the expense of downstream analysis (e.g. variant calling, genotype phasing, disease gene identification, read mapping, and genome assembly), biomedical researchers
have typically discarded quality scores altogether or turned to compression specialized for genomic sequence data [66, 67, 68, 43, 69]. High levels of quality score compression is usually achieved through lossy compression, meaning that maximum compression is achieved at the expense of the ability to reconstruct the original quality values [76, 70]. Due to decline in downstream accuracy, such methods are suboptimal for both transmission and indefinite storage of quality scores. To address these limitations, several methods exploit sequence data to boost quality score compression using alignments to a reference genome [67, 43, 74] or use raw read datasets without reference alignment [75]; however, reference-based compression requires runtime-costly whole-genome alignments of the NGS dataset, while alignment-free compression applies costly indexing methods directly to the read dataset. On the other hand, quality score recalibration methods, such as found in GATK [45], increase variant calling accuracy at the cost of significantly decreasing compressibility of the quality scores (Table 3.5). To our knowledge, no previous approach to Quartz simultaneously provides a scalable method for terabyte-sized NGS datasets and addresses the degradation of downstream genotyping accuracy that results from lossy compression [43].

To achieve scalable analyses, we take advantage of redundancy inherent in NGS read data. Intuitively, the more often we see a read sequence in a dataset, the more confidence we have in its correctness; thus, its quality scores are less informative and useful. However, for longer read sequences (e.g., >100bp), the probability of a read appearing multiple times is extremely low. For such long reads, shorter substrings (k-mers) can instead be used as a proxy to estimate sequence redundancy. By viewing individual read datasets through the lens of k-mer frequencies in a corpus of reads, we are able to ensure that ‘lossiness’ of compression does not deleteriously affect accuracy.

Quartz compresses quality scores by capitalizing on sequence redundancy. Compression is achieved by smoothing a large fraction of quality score values based on the k-mer neighborhood of their corresponding positions in the read sequences (Figure 3-1, left panel). We used the hypothesis that any divergent base in a k-mer likely corresponds to either a SNP or sequencing error; thus, we only preserve quality scores for
Figure 3-1: Visualization of Quartz algorithm. (a) A dictionary of common $k$-mers (green lines) in a corpus of NGS reads is generated. The dictionary is generated once for any species. (b) Each read sequence $R$ is broken up into overlapping supporting $k$-mers (purple). (c) Dictionary $k$-mers that are within one mismatch from the supporting $k$-mers are identified (mismatch positions in red). Top: Every position different from a dictionary $k$-mer is annotated as a possible variant (in red) unless covered by a dictionary $k$-mer corresponding to a different supporting $k$-mer, in which case it will nevertheless be marked for correction (in green; e.g. the second mismatch). Other covered positions are also marked for correction as high quality (green). Bottom: When two dictionary $k$-mers correspond to the same supporting $k$-mer, all mismatches are preserved, unless the mismatch position is covered by a dictionary $k$-mer corresponding to a different supporting $k$-mer as shown top. Uncovered bases are also annotated (blue). (d) Quality scores are smoothed and the scores of all high-quality positions (i.e. bases) are set to a default value. Scores of uncovered and possible variant loci are kept. (e) Quartz can fit into existing genotyping analysis pipelines as an additional processing step between acquisition of raw reads and mapping and genotyping.

probable variant locations and compress quality scores of concordant bases by resetting them to a default value. More precisely, frequent $k$-mers in a large corpus of NGS reads correspond to a theoretical consensus genome with overwhelming probability [60]; without having to do any explicit mapping, Quartz preserves quality scores at locations that potentially differ from this consensus genome. $k$-mer frequencies have been used to infer knowledge about the error content of a read sequence; in fact, many sequence-correction and assembly methods directly or indirectly make use of this phenomenon [77, 78], but not for quality score compression.

Unlike other quality score compression methods, Quartz simultaneously maintains genotyping accuracy while achieving high compression ratios, and is able to do so in
orders of magnitude less time. Compression is made possible by Quartz’s “coarse” representation of quality scores, which allows it to store quality scores at roughly 0.4 bits per value (from the original size of 8 bits in FASTQ format or 1.4 bits even after standard lossless text compression) (Tables 3.1 and 3.2). It is important to note that the compression ratio achieved by Quartz is primarily dependent on the conditional entropy of observing the read sequence, given the local consensus k-mer landscape. This advance is in contrast to lossless compressors, which can reproduce the original quality scores with perfect fidelity but are dependent on the entropy of the quality scores themselves, or to lossy methods that directly reduce the quality score entropy by quality score smoothing procedures [76, 3, 79].

Surprisingly, by taking advantage of the local consensus k-mer landscape, Quartz, while eliminating more than 95% of the quality score information, achieves improved genotyping accuracy compared with using the original, uncompressed quality scores as measured against a trio-validated (i.e. validated against parents’ genomes), gold-standard variant dataset for the NA12878 genome from the GATK “best-practices” bundle [45] (Figure 3-2). We applied both the GATK [45] and SAMtools [62] pipelines (Figure 3-1, right panel) to the compressed quality scores generated by Quartz on a commonly-used NA12878 benchmarking dataset from the 1000 Genomes Project [61] (Figure 3-2, Table 3.3). The genotyping accuracy based on Quartz’s compressed data consistently outperforms that based on the uncompressed raw quality scores as measured by the area under the receiver operating characteristic (ROC) curve (Table 3.4); for the experiments in Figure 3-2, Quartz compression decreases the number of false positives in the million highest quality variant calls by over 4.5% in several of the pipelines (Figure 3-2, top panels). While this improvement is most pronounced for SNP calls, indel-calling accuracy is also maintained, if not improved, by Quartz compression (Figure 3-3). This result emerges from the discovery through the application of Quartz that quality score values within an NGS dataset are implicitly encoded in the genomic sequence information with 95% redundancy, so often do not have to be stored separately. This improvement further indicates that compression achieved using Quartz reduces the noise in the raw quality scores, thus leading to bet-
ter genotyping results. Notably, removing all quality scores (by setting them all to a default value of 50) caused a substantial drop in genotyping accuracy (~5% decrease in relative ROC AUC) (Figure 3-4, left panel), indicating that retaining quality scores is necessary.

Quartz is also scalable for use on large-scale, whole-genome datasets. After a one-time construction of the $k$-mer dictionary for any given species, quality score compression is orders of magnitude faster than read mapping, genotyping, and other quality score compression methods (Table 3.5 and Figures 3-5 and 3-6). Additionally, Quartz is especially applicable for large-scale cohort-based sequencing projects, because its improvements in genotyping accuracy are particularly useful when samples have lower depths of sequencing coverage (e.g., $2\times$-$4\times$) (Figure 3-7).

Quartz alters quality scores and improves downstream genotyping accuracy, in common with some existing base quality score recalibration tools [45] that make use of human genome variation from population-scale sequencing [61], for example the GATK BaseRecalibrator. While currently available recalibration tools and Quartz both use distilled information from genome sequences, they differ in several important ways. Most importantly, quality score recalibration tools do not apply compression; in fact, GATK recalibration tends to greatly increase the amount of storage needed, while also requiring much more computing power (Table 3.5); Quartz avoids losing compressibility by using a single default replacement quality value (Figure 3-4, right panel). Furthermore, because recalibration tools such as the GATK BaseRecalibrator employ a list of known SNP locations, reads must first be mapped to the reference. As Quartz uses only $k$-mer frequencies and Hamming distances, it is possible to apply Quartz compression upstream of mapping, which is crucial for either genome assembly or mapping.

Quartz is the first scalable, sequence-based quality score compression method that can efficiently compress quality scores of terabyte-sized (or larger) sequencing datasets, thereby solving both the problems of indefinite storage and transmission of quality scores. Had our results merely replicated the genotyping accuracy of existing tools such as GATK and SAMtools, we would have still demonstrated order of
magnitude improved storage efficiency due to compression, at almost no additional computational cost. However, our results further suggest that a significant proportion of quality score data, despite having been thought entirely essential to downstream analysis, is less informative than the \(k\)-mer sequence profiles, and can be discarded without weakening (and nearly always improving) downstream analysis. Even with aggressive lossy data compression, we have shown that it is possible to preserve biologically important data.

A Quartz compression step can be added to almost any pre-existing NGS data processing pipeline. Quartz takes as input a FASTQ file (the standard format for read data) and outputs a smoothed FASTQ file, which can in turn be input into any compression program (e.g., BZIP2 or GZIP) for efficient storage and transmission, or any read mapper (e.g., BWA [49], Bowtie2 [41]). Further analysis steps such as variant calling (e.g., using SAMtools, GATK) can be carried out in the usual way. Our optimized and parallelized implementation of Quartz is available, along with a high-quality human genome \(k\)-mer dictionary. We also provide here preliminary results on how Quartz changes compression levels and variant-calling accuracy on an E. coli genome, indicating Quartz’s utility beyond human genomics (Figure 3-8).

3.2 Methods

3.2.1 Dictionary generation

To generate the dictionary, we compiled a list of commonly occurring 32-mers in a population-size corpus of sequencing reads. For the corpus, we used 194 FASTQ files taken from the 1000 Genomes Project [61], corresponding to paired-end reads from 97 human individuals (see 3.3.2). Read lengths in the dataset ranged from 91-103bp with a total depth of coverage of \(\sim 700\times\).

To construct the dictionary of \(k\)-mers memory efficiently, we modified the Misra-Gries approximate counting algorithm [80]:

111
Modified Misra-Gries:

1. Initialize a primary hash map \( A \).
   Initialize 65535 hash sets \( C_1, \ldots, C_{65535} \).
   Initialize the counter \( D = 0 \).

2. \textbf{update}(i):
   
   \begin{itemize}
   \item \textbf{if} \( i \in A \), remove \( i \) from \( C_{A[i]} \), increment \( A[i] \) (unless \( A[i] = 65535 \)),
     and insert \( i \) to the new \( C_{A[i]} \).
   \item \textbf{else if} \( |A| < m \), insert \((i, 1)\) into \( A \) and insert \( i \) into \( C_1 \).
   \item \textbf{else} Remove an arbitrary element \( j \) from \( C_D \)
     (while \( |C_D| = 0 \), increment \( D \) until nonempty).
     Remove \( j \) from \( A \).
     Insert \((i, D + 1)\) into \( A \) and \( i \) into \( C_{D+1} \).
   \end{itemize}

3. \textbf{query}(i):
   
   \begin{itemize}
   \item \textbf{if} \( i \in A \) \textbf{then} output \( A[i] - D \).
   \item \textbf{else} output 0
   \end{itemize}

Of note, in contrast to standard Misra-Gries, we use a collection of hash buckets instead of doubly linked lists to minimize cache misses. As we do not care about extremely high counts, we further arbitrarily limit the number of hash buckets to 65535. Memory usage is controlled by maintaining the invariant that the total number of \( k \)-mers stored never exceeds the memory-usage parameter \( m \). As with classic Misra-Gries, the approximation error is bounded by the decrement counter \( D \).

After partitioning the space of 32-mers into 256 subsets by the identity of the middle 4 bases, we ran the modified Misra-Gries in multiple passes on each of the subsets. By controlling the approximation ratio of the Misra-Gries algorithm, we guaranteed that all 32-mers in the dictionary occurred at least 200 times in the corpus and that all 32-mers in the corpus that occurred at least 240 times were included in the dictionary. In total, the dictionary contained 2,497,777,248 unique 32-mers.
3.2.2 Quality score compression

As input, our quality score compression algorithm Quartz requires the dictionary described above, consisting of commonly-occurring k-mers extracted from a population-sized read dataset. The dictionary is designed in such a way that any given read dataset can be mostly covered from these k-mers within a small Hamming distance, or number of mismatches. Using this dictionary, Quartz compresses the quality scores in any given read dataset by identifying k-mers from each read within a small Hamming distance from other k-mers in the dictionary. Any quality score value corresponding to a position that is concordant with at least one supporting k-mer is set to a default value, whereas the quality score value at any position that is divergent from all supporting k-mers is kept. This coarse representation greatly reduces the storage requirement for the quality scores of read datasets since the smoothed quality score values are substantially more compressible than the original values.

Our implementation was written in C++11 and compiled with GCC 4.7.2 with all relevant optimizations enabled. Because each read can be independently processed, Quartz is trivially parallelized using OpenMP, achieving speedups nearly linear to the number of cores used, up to the disk I/O bound.

3.2.3 Parameter selection

Efficiency concerns sufficiently dictate the parameters of k-mer length and Hamming distance that we have hard-coded optimal choices into the software. For the experiments presented, we selected the k-mer length parameter, $k$, to be 32bp. There are three main criteria we considered when selecting the k-mer length within the Quartz quality score compression framework:

1. $k$-mers should be long enough to ensure that the number of all possible $k$-mers is much larger than the number of unique $k$-mers in the genome, so as to ensure incidental collisions between unrelated $k$-mers are rare.

2. Since within the Quartz framework, $k$-mer neighbors are defined to be within a Hamming distance of 1, $k$-mers should also be short enough to allow the
probability that a k-mer contains more than one sequencing error to be low; this criterion is to ensure that k-mer sequences within reads originating from the same genomic region are highly likely to be detected as neighbors.

3. k-mer length should ideally be a multiple of four, since a 4bp length DNA sequence can be represented by a single byte.

A 32bp long k-mer satisfies all three of these criteria; it is represented by a single 64-bit integer, with a relatively low probability of containing more than one sequencing error with Illumina sequences, as well as resulting in few k-mer collisions. Experimental results demonstrating that other parameter values are inefficient can be found in Table 3.2.

For the experimental results presented, the default replacement value \((Q)\) for smoothed quality scores was selected as 50. In the datasets we studied, the average quality score excluding the special value of 2, which we did not modify, was at least 35. Recall that Quartz only replaces a quality score within a 32-mer if all Hamming neighbors agree on a position. Thus, assuming that all 32-mers within a read are variants of 32-mers in the dictionary, the method only incorrectly smooths a quality score if there are two read errors: one for the quality score and one at one of the other 31 locations. Then, as a first order approximation, the error probability for a smoothed location is \(31 \times 10^{-3.5} \times 10^{-3.5} < 10^{-5}\), justifying our choice of \(Q = 50\). However, this is a trivially adjusted parameter, and we have explored several replacement values in Figure 3-4.

### 3.2.4 Hamming neighbor search

As Hamming neighbors are defined as all k-mers in the dictionary that differ from a query k-mer by no more than 1 base substitution, we need to be able to quickly search for all \(3k + 1\) potential Hamming neighbors in the dictionary. The naive approaches of using a sorted list or hash tables suffer from the shortcomings of being respectively either CPU or memory inefficient. Additionally, both naive approaches incur many cache misses.
We instead used an approach inspired by locality sensitive hashing. Recall that an \((R, cR, P_1, P_2)\)-sensitive LSH family \(F\) of hash functions \(h : M \rightarrow S\) is defined if \(\forall p, q \in M, a\) uniformly random \(h \in F\) satisfies two conditions:

1. if \(||p - q|| \leq R\), then \(\text{Prob}(h(p) = h(q)) \geq P_1\), and

2. if \(||p - q|| \geq cR\), then \(\text{Prob}(h(p) = h(q)) \leq P_2\).

Note that projecting \(k\)-mers onto random \(\frac{k}{2}\)-length subsequences forms a \((1, \frac{1}{2}, 2c^{-1})\)-sensitive LSH hash family under the Hamming metric. Further note that any such projection \(h\) comes with an orthogonal projection \(h'\). Because the Hamming metric is discrete, all Hamming neighbors of a \(k\)-mer must match under one of the two projections. Thus, by double hashing, all Hamming neighbors can be found by looking in just two hash buckets, one for each hash function, giving much better cache-efficiency. Additionally, as genomic sequences are not chosen in an adversarial manner, instead of using random projections, projecting onto the front and back halves of the \(k\)-mer suffices in practice.

### 3.2.5 Memory requirements

As the dictionary is static and read-only, both hash tables can be efficiently implemented by sorting lexicographically first by the key (i.e. the projected half) and then by the rest, saving pointers to the first dictionary entry in each “bucket”, and then discarding the projected half of each \(k\)-mer—as all \(k\)-mers are only accessed through the bucket pointers, the hash key is implied. This scheme also allows for binary search within each bucket, further speeding up lookups. For the default \(k = 32\), we use an array of length \(2^{32}\) of unsigned 32-bit integers for the pointers and an array of unsigned 32-bit integers of length \(|D|\), the dictionary size, for the remaining half of the \(k\)-mers not specified by the pointers. Thus, total memory requirement for one such hash table is \((2^{32} + |D|) \cdot 4\) bytes, and since two hash tables are needed, \((2^{32} + |D|) \cdot 8\) bytes. For \(|D| = 2,497,777,248\) as for the human genome, Quartz requires \(~54\) GiB of memory.
3.3 Results

3.3.1 Experimental design

The existence of high quality trio-validated SNP calls makes NA12878 an ideal candidate for testing genotyping accuracy. We used the SRR622461 datasets from the 1000 Genomes Project [61] for the unmapped raw reads and the variant list CEU-Trio.HiSeq.WGS.b37.bestPractices.phased.hg19.vcf from the Broad GATK bundle as a gold standard for variant locations [45].

Quartz was used to generate smoothed FASTQ files with most quality scores reset to $Q$. To measure differences in entropy, both the original and modified quality scores were compressed with BZIP2, GZIP, 7z (PPMd), XZ (LZMA2), and FASTQZ [3]. The resulting file sizes were used to compute a rough estimate of the number of bits of storage needed per quality score. We ran BWA aln/sampe/samse (version 0.7.5a-r405) [49] and Bowtie2 (version 2.1.0) [41] with all default options to map the relevant FASTQ files. We used two state-of-the-art variant callers for estimating genotyping accuracy: GATK UnifiedGenotyper [45] and SAMtools mpileup [62]. We ran the GATK UnifiedGenotyper (version 2.4.7-g5e89f01) with default parameters. However, because Quartz resets base qualities to high, it does not interact well with the Base Alignment Quality filter [81]; thus, we ran SAMtools mpileup (version 0.1.18-dev r982:313) with all default options except for the Base Alignment Quality filter (varFilter option "-2 0").

To measure the relative downstream genotyping accuracy, we computed a rescaled receiver operating characteristic (ROC) curve on variant locations using the ROCR package [82]. Recall that variant callers do not simply give a list of variant calls. Rather, they give a list of variant calls along with variant call confidences. In order to provide a final list, some variant call confidence threshold must be chosen, at which point one can figure out the true and false positive ratio. The ROC curve is precisely the parameterized curve that results from varying that confidence threshold, and allows for comparing two methods without having to arbitrarily choose a single confidence threshold. However, in considering variant callers as binary classifiers, the
true negative rate of correctly called non-SNPs easily dwarfs everything else as most of the genome is called as not a SNP. As such, to compare two (or more) sets of variant call locations, we took the union of those call positions as the domain. The ROC curves were then computed against a gold standard classification of domain elements. Note that this rescaling implies that ROC curves are not comparable between different plots.

All timing benchmarks were run on a Debian GNU/Linux 6.0.8 system with dual Intel Xeon X5690 processors and 94 GiB RAM. However, to ensure comparability, we disabled parallelization, artificially limiting all programs to a single core unless specified otherwise.

3.3.2 Datasets

The dictionary used was generated by finding commonly occurring 32-mers in paired-end reads from recent 1000 Genomes Project FASTQ files from 97 individuals\textsuperscript{1} [83, 61]. We used two FASTQ files associated with each sample corresponding to the two mate pairs of all paired end reads for which both mate pairs passed filtering/quality control.

In addition to the NA12878 datasets we used, to ensure that our results are widely applicable, we also performed several analyses on NA19240, using datasets SRR794330 and SRR794336 for the raw unaligned reads. As a gold standard list of variant locations, we used Complete Genomics variant calls using CGAPipeline 2.2.0.26, generated on a high coverage (~ 80x) dataset using CGA Tools [84].

\textsuperscript{1}These FASTQ files were downloaded from ftp://ftp.1000genomes.ebi.ac.uk with the following sample names: NA06994, NA11930, NA11932, NA11933, NA11992, NA11994, NA12003, NA12046, NA12154, NA12155, NA12234, NA12249, NA12282, NA12283, NA12286, NA12340, NA12716, NA12717, NA12748, NA12750, NA12751, NA12761, NA12827, NA12829, NA12830, NA12842, NA12843, NA12878, NA12889, NA12890, NA18488, NA18501, NA18504, NA18528, NA18531, NA18533, NA18537, NA18572, NA18609, NA18611, NA18870, NA18881, NA18912, NA18966, NA18968, NA18969, NA18970, NA18971, NA18974, NA18975, NA18976, NA18978, NA18981, NA18991, NA19092, NA19093, NA19119, NA19131, NA19137, NA19138, NA19141, NA19143, NA19152, NA19160, NA19171, NA19175, NA19200, NA19204, NA19210, NA19222, NA19351, NA19649, NA19657, NA19658, NA19663, NA19664, NA19669, NA19670, NA19720, NA19764, NA19914, NA20357, NA20359, NA20362, NA20503, NA20507, NA20513, NA20514, NA20821, NA20900, NA21104, NA21120, NA21122, NA21123, NA21125, NA21127, and NA21137.
3.3.3 Timing and compression benchmarks.

In Table 3.1 we present the compression ratios of off-the-shelf compressors; Quartz smoothing reduces the amount of storage space needed by at least a factor of three. Additionally, also shown in the table is the performance of FASTQZ [3] on quality scores, a state-of-the-art FASTQ compressor that uses a clever lossless encoding of quality scores before compression, over which Quartz also demonstrates improvement.

There are two hard-coded parameters for Quartz compression, including the choice of k-mer length \( k = 32 \) and the restriction of Hamming neighbors to one substitution. We benchmark in Table 3.2 the effects of adjusting k-mer length and Hamming distance. For k-mer length, we benchmark only \( k = 16 \), as \( k = 64 \) requires infeasible memory requirements for most current machines, and for Hamming distance, we explore only distance 2, as CPU time increases exponentially with distance. As neither option performs well, a k-mer length of 32 and Hamming distance of 1 suffices for providing a good balance of performance criteria.

Table 3.1: Compression benchmarks (bits per quality score) on NA12878. After smoothing using Quartz, the number of bits needed to store a quality value was cut by over two thirds for post-processing with lossless compressors, including both off-the-shelf text compressors and FASTQZ, which is designed specifically for genomic data.

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Original</th>
<th>Quartz-smoothed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GZIP</td>
<td>1.7025</td>
<td>0.5043</td>
</tr>
<tr>
<td>BZIP2</td>
<td>1.3842</td>
<td>0.3564</td>
</tr>
<tr>
<td>7zip (PPMd)</td>
<td>1.2620</td>
<td>0.3676</td>
</tr>
<tr>
<td>XZ (LZMA2)</td>
<td>1.4353</td>
<td>0.4010</td>
</tr>
<tr>
<td>FASTQZ</td>
<td>1.1434</td>
<td>0.3089</td>
</tr>
</tbody>
</table>

Table 3.2: Timing (in CPU-seconds) and compression (in bits per quality value of the BZ2 file) benchmarks on NA12878 FASTQ files after adjusting k-mer length and Hamming distance. Both adjustments cause significant drops in both compression and speed.

<table>
<thead>
<tr>
<th></th>
<th>Bits/Q</th>
<th>CPU-seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quartz, k-mer length = 32, Hamming distance = 1</td>
<td>0.3564</td>
<td>2,696</td>
</tr>
<tr>
<td>Quartz, k-mer length = 16, Hamming distance = 1</td>
<td>1.7514</td>
<td>16,208</td>
</tr>
<tr>
<td>Quartz, k-mer length = 32, Hamming distance = 2</td>
<td>1.2844</td>
<td>140,128</td>
</tr>
</tbody>
</table>

Our implementation of Quartz costs a negligible amount of CPU time compared to mapping and variant calling. In Table 3.3 we give single-threaded timing benchmarks
for Quartz, mapping, and variant calling on NA12878 using the Bowtie2 [41] + GATK [85, 45, 86] and BWA [49] + SAMtools [62] pipelines presented in Figure 3-1, both with and without Quartz compression. Quartz is orders of magnitude faster than mapping and genotyping.

Also in Table 3.3 are presented timing benchmarks for off-the-shelf (lossless) compressors on just the quality scores, with and without Quartz compression. Of particular note is the fact that smoothing the quality scores using Quartz nearly doubles the speed of off-the-shelf compressors.

Table 3.3: Timing benchmarks (in seconds) on NA12878 taken from the Bowtie2 + GATK and BWA + SAMtools pipelines. Smoothing using Quartz is orders of magnitude faster than both mapping and variant calling steps. Furthermore, Quartz smoothing also significantly increases the speed of off-the-shelf lossless compressors.

<table>
<thead>
<tr>
<th></th>
<th>Quartz</th>
<th>Original Quality Scores</th>
<th>Quartz-smoothed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bowtie2</td>
<td>59,499</td>
<td>59,554</td>
<td></td>
</tr>
<tr>
<td>BWA</td>
<td>116,411</td>
<td>111,023</td>
<td></td>
</tr>
<tr>
<td>Variant calling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATK</td>
<td>23,721</td>
<td>22,261</td>
<td></td>
</tr>
<tr>
<td>SAMtools</td>
<td>24,121</td>
<td>24,190</td>
<td></td>
</tr>
<tr>
<td>Lossless compressors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GZIP</td>
<td>817</td>
<td>351</td>
<td></td>
</tr>
<tr>
<td>BZIP2</td>
<td>937</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>7zip (PPMd)</td>
<td>1,372</td>
<td>565</td>
<td></td>
</tr>
<tr>
<td>XZ (LZMA2)</td>
<td>12,600</td>
<td>6,551</td>
<td></td>
</tr>
</tbody>
</table>

Quartz improves downstream genotyping accuracy.

Here we demonstrate that for four different genotyping pipelines that we have tested, Quartz improves to varying degrees the downstream variant-calling accuracy over that of the uncompressed quality scores. Either with or without Quartz quality score compression with a default quality set to 50, NA12878 FASTQ files were aligned with either Bowtie2 [41] or BWA [49], and variant positions were called with either SAMtools mpileup [62] or the GATK UnifiedGenotyper [85, 45, 86].

2Because the FASTQZ code does not have an option to only compress quality scores without compressing sequence, we could not separate the two compression times and so have not included FASTQZ in the timing table, as it would be an unfair comparison for FASTQZ, which takes longer than off-the-shelf compressors, but has to compress sequences as well.
Resulting scaled ROC curves with corresponding AUC values are given in Figure 3-2.

Quartz preserves indel calling accuracy.

All of the other variant calling results are dominated by SNP-calling accuracy—GATK UnifiedGenotyper only gives SNP calls and although SAMtools mpileup gives other variant types, the calls made are predominantly SNPs. Because Quartz is designed with Hamming distance 1 in mind, substitutions are indeed the most appropriate variants for Quartz to handle. However, as Quartz ignores (and hence preserves) quality scores for k-mers greater than distance 1 from the dictionary, Quartz should not at all affect the accuracy of calling indels. In Figure 3-3, we show the preliminary results with and without Quartz compression for indel calls made using SAMtools mpileup. As expected, Quartz does not decrease indel calling accuracy, and even appears to slightly improve the AUCs for these two experiments, so Quartz at least preserves indel accuracy while achieving high compression.

Discarding all quality scores decreases variant-calling accuracy.

We additionally ran a control where we replaced all quality scores with 50, effectively discarding all of them. This causes a severe drop in genotyping accuracy using the Bowtie2 + GATK pipeline (Figure 3-4, left panel), demonstrating that retaining some of the quality scores is necessary.

Varying the default replacement quality value can slightly affect the downstream variant calling accuracy.

While all the results presented here use a replacement quality value $Q = 50$, this parameter is trivially adjusted and for appropriately high values, has no effect on either compression or CPU-time. Indeed, as suggested in the Online Methods, the optimal choice for this parameter can be related to the average quality value of the dataset at hand. In Figure 3-4 (right panel), we explore the result on downstream variant calling accuracy from varying that value. While the choice of $Q = 40$ was
Figure 3-2: Downstream SNP calling before and after Quartz compression on four different genotyping pipelines for NA12878, using BWA and Bowtie2 as mappers and GATK UnifiedGenotyper and SAMtools as genotype callers. Additionally, we give the number of false positives in the million highest quality variant calls for each pipeline.
slightly worse, $Q = 45$ and $Q = 50$ were nearly indistinguishable. A more careful examination of the slight differences reveals that $Q = 50$ results in slightly more false positives, accounting for the marginal difference between the latter two in AUC.

Quartz out-performs existing lossy compression methods in terms of compression ratio, speed, and genotyping accuracy.

Here we show that while maintaining superior compression levels and improving variant-calling accuracy, Quartz offers an order of magnitude improvement in speed compared to other state-of-the-art lossy compression methods in the literature. Quality scores in the FASTQ files were first compressed using one of Quartz, QualComp [76], or Janin et al [75], aligned with Bowtie2 [41], and then GATK UnifiedGenotyper [85, 45, 86] was used for genotyping. Resulting scaled ROC curves, total single-threaded processing time for compression/decompression, and resulting size of quality scores are given in Figures 3-5 and 3-6. As Quartz and Janin et al. only alter quality scores in-place to improve compressibility, they were compressed using Bzip2, whereas QualComp has its own compressed file format. Quartz was run with default quality 50. QualComp was run with bits per read set to 60. The method of Janin et al. was
Figure 3-4: Changes in ROC curves when quality score values are removed or assigned different default values within the Quartz algorithm. (Left panel) Discarding all quality scores results in much lower genotyping accuracy. Note that the impact of ROC curve rescaling is most apparent in this figure because of the large numbers of false positives called when all quality scores are discarded. (Right panel) Downstream variant calling before and after Quartz compression on the Bowtie2 + GATK UnifiedGenotyper pipeline for NA12878, with varying replacement quality values. Note that the AUC values are not comparable between left and right panels, due to ROC curve scaling based on the union SNP call set. The red and blue ROC curves are shifted towards left in the left panel, due to large number of false positive calls made by the no quality score run.
run with parameters c=0, s=2, r=39.

![NA12878 - Bowtie2 + GATK](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Bits</th>
<th>Time (s)</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncomp.</td>
<td>8</td>
<td>N/A</td>
<td>0.8048</td>
</tr>
<tr>
<td>Quartz</td>
<td>.356</td>
<td>2.7</td>
<td>0.8157</td>
</tr>
<tr>
<td>QualComp</td>
<td>.594</td>
<td>33.3</td>
<td>0.8053</td>
</tr>
<tr>
<td>Janin et al.</td>
<td>.538</td>
<td>164.7</td>
<td>0.8019</td>
</tr>
</tbody>
</table>

**Figure 3-5:** Quartz compared to QualComp and the method of Janin et al. on NA12878, showing bits used per quality score value, time in seconds and area under the curve. Quartz is able to simultaneously improve compression ratio, speed, and genotyping accuracy.

**Figure 3-5:** Quartz compared to QualComp and the method of Janin et al. on NA12878, showing bits used per quality score value, time in seconds and area under the curve. Quartz is able to simultaneously improve compression ratio, speed, and genotyping accuracy.

![NA12878 - Bowtie2 + GATK](image)

**The improvement in accuracy from Quartz is consistent across chromosomes.**

Although the improvement in accuracy over the uncompressed is easily visible for NA12878, it is less so for NA19240, likely because the gold standard variant calls being used are not as well characterized. However, by demonstrating consistent improvements in variant calling accuracy for all chromosomes, our assertion of increased accuracy is substantiated in Table 3.4.

**Quartz shows greater improvement in genotyping accuracy at lower coverage read datasets.**

The reads for NA12878 shown in all figures except for Figure 3-7 had a total depth-of-coverage of about 6. For Figure 3-7, the reads were subsampled, preserving mate pairs, and the effect on downstream variant calling was assessed. Although the ROC curves and absolute AUCs are not comparable because of rescaling, we can compare relative differences in AUC. Doing so, we find that Quartz seems to show a greater
Table 3.4: NA12878 and NA19240 ROC AUC values for original uncompressed vs Quartz, broken down by chromosome, after using the Bowtie2 + GATK Unified Genotyper pipeline.

<table>
<thead>
<tr>
<th>Chrom.</th>
<th>NA12878</th>
<th></th>
<th>NA19240</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original AUC</td>
<td>Quartz AUC</td>
<td>Quartz Improv.</td>
<td>Original AUC</td>
</tr>
<tr>
<td>1</td>
<td>0.7893</td>
<td>0.8105</td>
<td>0.0212</td>
<td>0.7255</td>
</tr>
<tr>
<td>2</td>
<td>0.7912</td>
<td>0.8140</td>
<td>0.0228</td>
<td>0.7286</td>
</tr>
<tr>
<td>3</td>
<td>0.7878</td>
<td>0.8110</td>
<td>0.0232</td>
<td>0.7227</td>
</tr>
<tr>
<td>4</td>
<td>0.7940</td>
<td>0.8171</td>
<td>0.0231</td>
<td>0.7012</td>
</tr>
<tr>
<td>5</td>
<td>0.7922</td>
<td>0.8125</td>
<td>0.0203</td>
<td>0.7209</td>
</tr>
<tr>
<td>6</td>
<td>0.7798</td>
<td>0.8036</td>
<td>0.0238</td>
<td>0.7183</td>
</tr>
<tr>
<td>7</td>
<td>0.7829</td>
<td>0.8063</td>
<td>0.0235</td>
<td>0.7112</td>
</tr>
<tr>
<td>8</td>
<td>0.7818</td>
<td>0.8017</td>
<td>0.0199</td>
<td>0.7157</td>
</tr>
<tr>
<td>9</td>
<td>0.7760</td>
<td>0.7980</td>
<td>0.0220</td>
<td>0.7302</td>
</tr>
<tr>
<td>10</td>
<td>0.7778</td>
<td>0.7993</td>
<td>0.0215</td>
<td>0.6959</td>
</tr>
<tr>
<td>11</td>
<td>0.7720</td>
<td>0.7954</td>
<td>0.0234</td>
<td>0.7148</td>
</tr>
<tr>
<td>12</td>
<td>0.7799</td>
<td>0.8006</td>
<td>0.0207</td>
<td>0.7270</td>
</tr>
<tr>
<td>13</td>
<td>0.7923</td>
<td>0.8151</td>
<td>0.0228</td>
<td>0.7088</td>
</tr>
<tr>
<td>14</td>
<td>0.7855</td>
<td>0.8059</td>
<td>0.0204</td>
<td>0.7245</td>
</tr>
<tr>
<td>15</td>
<td>0.7841</td>
<td>0.8075</td>
<td>0.0234</td>
<td>0.7362</td>
</tr>
<tr>
<td>16</td>
<td>0.7601</td>
<td>0.7889</td>
<td>0.0282</td>
<td>0.7078</td>
</tr>
<tr>
<td>17</td>
<td>0.7738</td>
<td>0.7986</td>
<td>0.0248</td>
<td>0.7216</td>
</tr>
<tr>
<td>18</td>
<td>0.7823</td>
<td>0.8064</td>
<td>0.0242</td>
<td>0.7244</td>
</tr>
<tr>
<td>19</td>
<td>0.7407</td>
<td>0.7687</td>
<td>0.0280</td>
<td>0.6965</td>
</tr>
<tr>
<td>20</td>
<td>0.7687</td>
<td>0.7930</td>
<td>0.0243</td>
<td>0.7354</td>
</tr>
<tr>
<td>21</td>
<td>0.7367</td>
<td>0.7696</td>
<td>0.0329</td>
<td>0.6588</td>
</tr>
<tr>
<td>22</td>
<td>0.7413</td>
<td>0.7703</td>
<td>0.0290</td>
<td>0.7174</td>
</tr>
<tr>
<td>X</td>
<td>0.8118</td>
<td>0.8293</td>
<td>0.0176</td>
<td>0.7470</td>
</tr>
</tbody>
</table>

Mean improv.: 0.0235 ± 0.0007  Mean improv.: 0.0044 ± 0.0003
improvement at lower coverage, while it must be noted that in absolute terms, the improvement that comes from higher coverage is greater than the improvement Quartz compression provides.

**Quartz can be effectively applied to the *Escherichia coli* genome.**

All of the other results presented in this chapter have been on the human genome, which is naturally of particular interest. However, while new k-mer dictionaries need to be generated for each species, Quartz can be successfully applied to compressing quality scores from other species, as demonstrated here in preliminary results with the K-12 MG1655 and DH10B strains of *E. coli*. For this experiment, we took paired-end sequencing reads of *E. coli* K-12 strain MG1655 using Illumina Genome Analyzer II from the European Nucleotide Archive [87] Study: ERP000092 and of *E. coli* K-12 strain DH10B using Illumina MiSeq, downloaded from the Orione database [88]. The reference used was U00096.fna from the NIH Genbank for MG1655, and a truth set for DH10B as compared to MG1655 was generated by sampling one million 1000-mer sequences from the DH10B reference CP000948.fna and running Bowtie2 and GATK Unified Genotyper on it. The dictionary was generated from the corpus of reads.
Figure 3-7: ROC curves for subsampled lower-coverage versions of the NA12878 dataset. Relative improvements in AUC given by Quartz using the Bowtie2 + GATK UnifiedGenotyper pipeline are given in the table.
consisting of just the MG1655 and DH10B reads described above, such that all 32-mers with frequency at least 101 were included and only 32-mers with frequency at least 100 were included. Note that the depth-of-coverage for the corpus was ~850x.

As Quartz performs best on low-coverage datasets, we subsampled the DH10B reads down to ~9.5x depth-of-coverage for the analysis. In Figure 3-8, we give SNP-calling accuracies for the uncompressed, Quartz-compressed, Qualcomp-compressed, and Janin et al-compressed quality scores, after attempting to match compression ratios. Note that this was possible for Qualcomp, which has an explicit bits/read parameter, but not for the method of Janin et al, so we chose a parameter set allowing the method of Janin et al additional bits to use.

As a caveat, at higher depth-of-coverage, the improvement of Quartz over the uncompressed quality scores is reduced or sometimes even lost, in marked contrast to the human results, which always display improvement. This anomaly is likely due to the use of a low-quality dictionary; whereas the human corpus consisted of 97 different individuals, this corpus contains only 2 strains of *E. coli*. As less variation is encoded in the k-mer landscape of this corpus, Quartz thus has less information with which to improve variant calling accuracy. However, these preliminary results do suggest that Quartz is applicable to non-human genomes, though with the caveat that results will only be as good as the dictionary used.

**Quality score recalibration decreases compressibility.**

Although there are some superficial similarities to quality score recalibration methods, in that both they and Quartz use k-mer profiles and improve downstream variant calling accuracy, we demonstrate here in Table 3.5 the major difference: Quartz is a compression algorithm, and decreases space requirements, whereas the GATK [45] recalibration increases storage space required. We demonstrate this on the quality scores for NA12878 after mapping by Bowtie2 [41]. As a note, because the GATK recalibration is performed post-mapping and sorting by read position, the compression ratios differ slightly from those given in Table 3.1 for unmapped reads. Although Quartz is performed pre-mapping, here we give the compression results after mapping
Figure 3-8: Quartz compared to QualComp and the method of Janin et al. on a subsampling of reads from E. coli strain K12-DH10B to 9.5x depth-of-coverage. The method of Janin et al was run with parameters c=0, s=2, r=39, QualComp was run with bits per read set to 39.3, and Quartz was run with default replacement value Q50. For reference, performance after lossless compression by BZIP2 is also included with the results on the original scores.

to ensure comparability to the GATK recalibration. Finally, total time required for the listed steps is given, demonstrating that Quartz compression is much faster than GATK recalibration.

Table 3.5: Time requirements and effect on storage of Quartz compression and GATK recalibration on NA12878, measured by CPU-time and BZ2 file size. Of note, recalibration significantly increases storage size. Furthermore, recalibration is much slower than Quartz compression.

<table>
<thead>
<tr>
<th>Method</th>
<th>Bits</th>
<th>Time (s)</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>8</td>
<td>0</td>
<td>0.9617</td>
</tr>
<tr>
<td>BZIP2</td>
<td>3.039</td>
<td>7</td>
<td>0.9617</td>
</tr>
<tr>
<td>Quartz</td>
<td>0.2587</td>
<td>7</td>
<td>0.9672</td>
</tr>
<tr>
<td>QualComp</td>
<td>0.2585</td>
<td>153</td>
<td>0.9404</td>
</tr>
<tr>
<td>Janin et al.</td>
<td>0.6477</td>
<td>383</td>
<td>0.8952</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>CPU-time</th>
<th>Bits/Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0</td>
<td>1.4159</td>
</tr>
<tr>
<td>GATK recalibrated</td>
<td>22,686</td>
<td>1.9854</td>
</tr>
<tr>
<td>Quartz compressed</td>
<td>2,696</td>
<td>0.3601</td>
</tr>
</tbody>
</table>
Chapter 4

Polyploid Haplotype Reconstruction Using NGS Datasets

4.1 Overview

While human and other eukaryotic genomes typically contain two copies of every chromosome, plants, yeast and fish such as salmon can have strictly more than two copies of each chromosome. By running standard genotype calling tools, it is possible to accurately identify the number of "wild type" and "mutant" alleles (A, C, G, or T) for each single-nucleotide polymorphism (SNP) site. However, in the case of two heterozygous SNP sites, genotype calling tools cannot determine whether "mutant" alleles from different SNP loci are on the same or different chromosomes (i.e. compound heterozygote). While the former would be healthy, in many cases the latter can cause loss of function; it is therefore necessary to identify the phase (phasing), the copies of a chromosome on which the mutant alleles occur, in addition to the genotype (Figure 4-1). This necessitates efficient algorithms to obtain accurate and comprehensive phase information directly from the next-generation-sequencing read data in higher ploidy species.
Figure 4-1: Loss of function in different polyplotypes of a sample pentaploid genome. As the loss of function is often determined by whether a healthy copy of a gene exists, knowing the genotype vector is sufficient if there is a single SNP site. In the case of two SNP sites however, the genotype vector cannot be used to unambiguously determine loss of function, and phasing is required.

Various sources of information can be utilized for the computational identification of an individual’s diplotype/polyplotype: pedigree (e.g. trio-based phasing) [89, 90, 91], population structure of variants (e.g. phasing by linkage disequilibrium) [91, 92, 93, 94] and more recently by identity-by-descent in unrelated individuals [95, 96], as well as sequencing read datasets [97, 98, 99, 100, 101]. Among these approaches, methods for sequence-based haplotype phasing are the only viable approach for haplotype phasing on a single individual member of a species (assuming homologous chromosomes are sequenced together), as other approaches either require family members or a population. For an individual diploid genome, the problem of reconstructing the diplotype using sequence information, the diploid phasing problem, is equivalent to the identification of the sequence of alleles on either parental haplotype. If this sequence is correctly inferred, then the other haplotype will automatically carry the corresponding opposite alleles (reference or alternative). Solving an error-free version of the diploid haplotype reconstruction problem is straightforward: the haplotype of each connected (by reads) component of heterozygous SNPs can be obtained by propagating allele information within reads. In reality, however, sequencing errors as well as false read mappings cause conflicts within sequence information, requiring a mathematical formulation of the haplotype reconstruction problem. Among various formulations suggested for this problem, the most commonly used is an NP-hard minimum error correction (MEC) definition [102, 103], which aims to identify
the smallest set of nucleotide changes required within mapped fragments that would allow a conflict-free separation of reads into two separate homologous chromosomes (or a bipartite separation of the fragment conflict graph). Some of the solutions proposed for this problem include: HapCUT [97], an algorithm for optimizing MEC score based on computing max-cuts of the fragment graph; Fast Hare [104], a heuristic that clusters reads into two sets in a greedy fashion, and HapCompass [98], a spanning tree based approach for minimizing fragment conflicts.

Unlike diploid genomes, computational identification of common chromosomal variants in polyploid genomes using sequencing data has received little attention, except in the pioneering work of Aguiar and Istrail [96]. Polyploidy studies are of importance as they allow a comprehensive investigation of variants within plant, fish, and yeast genomes and help understand mechanisms of eukaryotic evolution. However, haplotype reconstruction in polyploid genomes is fundamentally more complex, even in the error-free version of the problem (without sequencing errors or false read mappings). Due to the newness of the NGS-based biological research in polyploid genomes, the mathematical foundations of the polyploid phasing problem have not yet been established. The solution proposed by Aguiar and Istrail for single individual polyplotyping problem is based on phasing all possible SNP loci pairs independently while further consolidating this information in a separate stage in order to infer a set of haplotypes.

Diploid phasing methods focus on a given list of heterozygous variants that are guaranteed to contain a single reference allele, as well as an alternative allele (assuming all heterozygous loci are bi-allelic). In contrast, in the polyploid phasing problem, there is no such guarantee of a single type of heterozygous SNP. Each heterozygous locus for a k-ploid chromosome can potentially contain from 1 up to k – 1 alternative alleles within the heterozygous loci, significantly increasing the complexity of the phasing problem in comparison to the diploid case. Furthermore, in a diploid phasing setting, there are always two possible options for phasing a pair of SNP loci, regardless of what other SNPs they are phased with. These two options can be thought as parallel (alternative allele pairs and reference allele pairs are matched within themselves).
or switched (each alternative allele is matched with the other reference allele). These two options are no longer relevant when the genome contains more than two copies of each chromosome, due to the fact that there are up to \( k! \) options when merging a phased haplotype block with another.

This chapter introduces a maximum-likelihood formulation of the polyploid full haplotype reconstruction problem and present a haplotype assembly algorithm, HapTree, which concurrently performs SNP-pair phasing and full haplotype assembly based on a probabilistic framework. We observe that, on simulated polyploid data, HapTree substantially improves the phasing capabilities and performance of any existing program. Because real polyploid data is hard to come by, we also evaluate HapTree on real human diploid data and find that, when compared to the more accurate trio-based data as the ground truth [83], HapTree significantly reduces the number of switch errors, while remaining on par in terms of MEC score over existing single-individual haplotype assembly methods for diploid genomes. We also introduce a relative likelihood (RL) score definition for annotation-free evaluation of phasing quality for polyploid haplotype assembly as an alternative to MEC score. Using simulated polyploid sequencing datasets, we demonstrate that RL-score performs significantly better at capturing haplotype assembly quality than MEC-score as ploidy increases.

4.2 Methods

The HapTree pipeline is designed to perform phasing and full haplotype assembly of a single genome. The key component of HapTree is a relative likelihood function which measures the concordance between the aligned read data and a given haplotype phase under a probabilistic model that also accounts for possible sequencing errors. To identify a phasing solution of maximal likelihood, HapTree finds a collection of high-likelihood solutions for phases of the first \( m \) SNP loci and extends those to high likelihood phases of the first \( m + 1 \) SNP loci, for each incremental \( m \). In each step, HapTree maintains only the set of likely partial phases to be extended in next steps.
Finally, a phase of maximal likelihood for all loci is obtained after the extension of the last SNP locus.

Broadly speaking, HapTree aims to discover the best, or maximum likelihood, haplotype based on the read data available. Theoretically, one could enumerate all possible haplotypes, compute the likelihood of each being the true haplotype (using formulas described below), and choose the most likely one; in most cases this approach is intractable as there are exponentially many possible haplotypes. HapTree therefore has a variety of ways of trimming down the solution set from all possible haplotypes to a much smaller set of more likely solutions, making the problem tractable. It does so by taking an inductive approach, generating a collection of likely phasing solutions for the first two SNPs in the genome, and then extending those to phasing solutions of the first three SNPs, and those to the first four SNPs, and so on. When extending any particular solution, HapTree chooses (based on computing likelihoods) how the alleles of the newly added SNP may be assigned to chromosomes; it includes only those assignments that are sufficiently likely. Additionally, if HapTree finds after extending all solutions to include the next SNP that there are too many likely solutions, it throws the worst (least likely) solutions away. Upon including all SNPs to be phased, HapTree randomly chooses a solution of maximum likelihood from amongst the solutions it has found.

An implementation of our HapTree is available for download at:

http://groups.csail.mit.edu/cb/haptree/

4.2.1 Definitions and notation

This section describes the problem of sequence-based polyploid haplotype assembly and provides basic technical notation that will be useful for describing the HapTree method. We assume for now that each SNP locus to be phased is bi-allelic (i.e. contains only two possible alleles, one being the reference allele). We further assume that for each SNP locus s, the genotype of s is known and is equal to the number of chromosomes carrying the alternative allele (denoted by \(g(s)\)). If \(k\) denotes the ploidy, \(g(s)\) can range from 1 to \(k - 1\) for heterozygous loci s.
We denote the sequence of observed nucleotides of a fragment simply as a “read” (independent from single/paired-end reads). The set of all reads is denoted as \( R \). We define a read \( r \in R \) as a vector with entries \( r[i] \in \{0, 1, -\} \) where a 0 denotes the reference allele, a 1 the alternative allele, and a \(-\) indicates one of two possibilities: First, that the read does not overlap with the corresponding SNP locus, or second, that neither the reference nor alternative allele is present and hence there must be a read error. A read \( r \in R \) contains a SNP \( s \) if \( r[s] \neq - \). A read can also be represented as a dictionary or mapping with keys the positions (from amongst the SNPs to be phased) of SNP loci it contains and values of either reference allele or alternative allele, represented by 0 and 1 respectively (e.g. \( r = \{3:0, 4:1, 5:0, 8:1, 9:1\} \)). As current sequencing technologies generate read data with a certain rate of sequencing errors, some of the positions within a read likely contain false nucleotide information. Among these erroneous bases, unless they are located at SNP loci and contain opposite allele information, we ignore them by representing them with \(-\), and thus keep only confounding sequencing errors that can affect phased haplotype results. For each read \( r \) and for each SNP locus \( s \), we assume an error rate of \( \varepsilon_{r,s} \) and a probability of opposite false allele information \( r[s] \) is equal to \( \varepsilon_{r,s} = \frac{\varepsilon_{r,s}}{1 - \frac{3}{2}\varepsilon_{r,s}} \). We modify this error rate by a factor of two-thirds because conditional on there being an error, the error is equally likely to be any of the three other alleles. Two of the three of these alleles are neither the reference nor the alternative allele and thus we know that an error has been made in this case. Therefore, two-thirds of the time the erroneous alleles produced are known as such and may be thrown out, leaving a true error only one-third of the time. We represent these error rates as matrices \( \varepsilon, \varepsilon \).

Upon the set of SNP loci \( S \) and read set \( R \); we define a read graph, \( G(S, R) \), such that there is a vertex for each SNP locus \( s \in S \) and an edge between any two vertices \( s_1, s_2 \) if there is some read containing both \( s_1 \) and \( s_2 \); equivalently if \( \exists r \in R, r[s_1] \neq - \land r[s_2] \neq - \). Without loss of generality, we assume that \( G(S, R) \) is connected; otherwise each connected component can be processed independently.
**Vector set**

A \( k \)-ploidy phase of \( n \) SNPs with genotypes \( \{g(s)\} \) is a tuple of \( k \) vectors (not necessarily distinct) \( (v_1, ..., v_k) \subset \{0,1\}^n \) satisfying the genotype allele counts property, that is: \( v_1[s] + v_2[s] + ... + v_k[s] = g(s) \) for all \( s \in \{1, 2, ..., n\} \). We will refer to this collection as a vector set and we think of each vector as a row vector.

We can build a phase by selecting a permutation of the alleles present for each SNP locus \( s \). Note that the number of distinct permutations, \( C(s) \), is strictly dependent on the genotype of the SNP and in the diploid bi-allelic case is equivalent to selecting the chromosomes containing the alternative alleles, hence

\[
C(s) = \binom{k}{g(s)} = \frac{k!}{g(s)!(k - g(s))!}
\]

For example, let \( k = 4 \), then \( g(s) \in \{1, 2, 3\} \). We enumerate the possible permutations below and include an example tetraploid genome.

\[
\begin{align*}
0001 & \quad 000111 & \quad 0111 \\
0010 & \quad 011001 & \quad 1011 \\
0100 & \quad 101010 & \quad 1101 \\
1000 & \quad 110100 & \quad 1110 \\
\end{align*}
\]

14 possible permutations of tetraploid bi-allelic heterozygous SNPs, shown as columns. Each column corresponds to a heterzygous SNP locus with 1, 2, or 3 alternative alleles

\[
\begin{align*}
00011010 & \\
01110011 & \\
10111100 & \\
10000111 & \\
\end{align*}
\]

\{\(00011010\), \(01110011\), \(10111100\), \(10000111\)\}

A sample tetraploid genome and its corresponding vector set.

The sample tetraploid genome featured above has a genotype vector: \([2, 1, 2, 3, 2, 2, 3, 2]\); recall this counts the number of alternative alleles present at each SNP site. For any SNP \( s \), let \( P_s \) denote the set of distinct allele permutations at SNP locus \( s \).
Throughout we are indifferent to the order of each chromosome, with this in mind we can see that the total number of phases is bounded below by $\frac{1}{k!} \prod C(s)$.

**Likelihood of a phase**

We formulate the haplotype reconstruction problem as identifying the most likely phase(s) given the read data $R$, all SNP loci $S$, as well as their genotypes, and sequencing error rates $\varepsilon$. We assume the sequencing errors are independent of each other, that is for all $r \in R$ and all $s \in r$, that $\{r[s]\}$ are independently correct with probabilities $(1 - \varepsilon_{r,s})$ and incorrect with probabilities $\varepsilon_{r,s}$. Let $\varepsilon$ be a matrix containing all of these probabilities: $\{\varepsilon_{r,s}\}$. Given a vector set, $V_{set}$, corresponding to a phase, $R$, and $\varepsilon$; the likelihood of the phase is determined by:

$$P[V_{set}|R, \varepsilon] = \frac{P[R|V_{set}, \varepsilon] P[V_{set}|\varepsilon]}{P[R|\varepsilon]}.$$ (4.1)

As $P[R|\varepsilon]$ depends only on $\varepsilon$ and the read set $R$, it is therefore the same across all vector sets. Hence, we define a relative likelihood measure (RL) as

$$RL[V_{set}|R, \varepsilon] = P[R|V_{set}, \varepsilon] P[V_{set}|\varepsilon].$$

As for $P[V_{set}|\varepsilon]$, there are several ways this can be modeled depending on the situation. For polyploid simulated data, we can assume that $P[V_{set}|\varepsilon]$ is equal for almost all vector sets, excluding ones containing duplicate vectors. Let $M = \{m_1, m_2, ...\}$ be the set of the multiplicities in $V_{set}$; for example, if $V_{set} = \{0001, 0010, 1100, 0001, 0010\}$ then $M = \{2, 2, 1\}$. The probabilities $P[V_{set}|\varepsilon]$ will differ multiplicatively by multinomial coefficients $\binom{k}{m_1, m_2, ...} = \frac{k!}{m_1! m_2! ...}$.

Specifically:

$$P[V_{set}|\varepsilon] = \frac{k}{\prod C(s)}.$$

For real diploid data, there will never be duplicate vectors. To model $P[V_{set}|\varepsilon]$, we might assume that since mutations tend to occur together, adjacent SNP sites are more likely to be phased in parallel (00) or (11) than switched (01) or (10). Let
\( V = (v, v') \) and let \( P(V) \) denote the number of adjacent SNPs that are parallel in \( v \) and \( S(V) \) the number of adjacent SNPs that are switched in \( v \) (we must only consider \( v \) as it determines \( v' \)). For example, if \( V = ((00010111000), (11101000111)) \), then \( P(V) = 6 \) and \( S(V) = 4 \). For some \( p > .5 \) (denoted as parallel bias) and \( q = 1 - p \), we model this vector set probability as

\[
P[V_{\text{set}} | \varepsilon] = p^{P(V_{\text{set}})} q^{S(V_{\text{set}})}
\]

Finally, we consider \( P[R|V_{\text{set}}, \varepsilon] \). For a given \( r \in R \) and \( v \in V_{\text{set}} \), let \( A(r, v) \) and \( D(r, d) \) denote the positions of SNP loci where \( r \) and \( v \) agree and disagree respectively. For example, if \( r = (-, -, 1, 0, 1, -, -, 1, 0) \) and \( v = (1, 0, 0, 1, 1, 0, 1, 0, 1, 0) \), then \( A(r, v) = (5, 8, 9) \) and \( D(r, v) = (3, 4) \) We may now compute the desired probability, that is:

\[
P[R|V_{\text{set}}, \varepsilon] = \prod_{r \in R} P[r|V_{\text{set}}, \varepsilon]
\]

\[
P[r|V_{\text{set}}, \varepsilon] = \frac{1}{k} \sum_{v \in V_{\text{set}}} \left( \prod_{s \in A(r, v)} (1 - \varepsilon_{r,s}) \prod_{s \in D(r, v)} \varepsilon_{r,s} \right)
\]

The goal of our haplotype reconstruction problem is to find the vector set(s) maximizing the product \( P[R|V_{\text{set}}, \varepsilon] P[V_{\text{set}}|\varepsilon] \), equivalently \( RL[V_{\text{set}}|R, \varepsilon] \). However, as the number of possible phases is on the order \( k^n \), checking all of these is intractable. Our solution is based on finding high likelihood phases for the first \( m+1 \) SNPs, conditioned on a collection of high likelihood phases for the first \( m \) SNPs.

**Semi-reads and sub-reads**

To properly describe our method we must first define the semi-reads of a SNP locus \( s \) and the sub-reads of a subset \( S' \subset S \).

**Semi-reads.** To form the set of semi-reads of \( s \), denoted \( SR(s) \), include each read \( r \in R \) that contains both \( s \) and some \( s' < s \) (\( s' \) is upstream of \( s \)) and ignore all information from \( r \) on SNPs \( s'' > s \) (\( s'' \) is downstream of \( s \)). Suppose the set of reads is:
The corresponding semi-reads for each SNP locus would be:

1 → None
2 → \{1:1, 2:1\}
3 → \{1:1, 2:1, 3:1\}
4 → \{1:1, 2:1, 3:1, 4:1\} \{3:1, 4:1\}
5 → \{3:1, 4:1, 5:0\} \{4:0, 5:1\} \{4:0, 5:1\}
6 → \{3:1, 4:1, 5:0, 6:0\} \{4:0, 5:1, 6:1\} \{4:0, 5:1, 6:1\} \{5:0, 6:0\} \{5:1, 6:1\}
7 → \{4:0, 5:1, 6:1, 7:0\} \{5:0, 6:0, 7:1\} \{5:1, 6:1, 7:0\}

Sub-reads. The sub-reads of \(S' \subset S\), denoted \(R(S')\), are obtained by, for each \(r \in R\), removing all keys \(s \in S \setminus S'\) to form \(r'\), and then adding \(r'\) to \(R(S')\) if the length of \(r'\) is at least 2. Alternatively, \(R(S')\) corresponds to the set of reads relevant to the problem of only phasing \(S'\). Continuing with the example above, if \(S' = \{1, 2, 3, 4, 5\}\), then

\[
R(S') = \{\{1:1, 2:1, 3:1, 4:1\}, \{3:1, 4:1, 5:0\}, \{4:0, 5:1\}, \{4:0, 5:1\}\}.
\]

### 4.2.2 HapTree algorithm

Our main approach to solving the single individual polyploid haplotype assembly problem is by finding highly probable solutions on \(m\) SNPs and extending those to highly probable solutions on \(m+1\) SNPs. Our algorithm has two fundamental parts: branching and pruning. For each connected component of the ReadGraph, \(G(S, R)\), we inductively generate a collection of high likelihood phases on the first \(m\) SNPs. For each of these phases, we branch them to phases on \(m+1\) SNPs by considering all possible orderings of alleles for position \(m+1\) and including branches for those which occur with probability above a certain threshold. After doing so, we prune the tree of phases by removing all leaves that occur with probability sufficiently less than the most probable leaf. We discuss both parts in more detail below. We note that although a dynamic programming algorithm can be directly applied to infer the best solutions under HapTree’s likelihood model, we instead developed a beam search approach.
algorithm, which is faster than exact dynamic programming but with nearly identical empirical performance.

**Extension**

We first describe how to extend an existing a haplotype assembly $H$ on $m \geq 0$ SNPs onto the $(m + 1)^{th}$ SNP $s$. Recall the set of permutations of $s$ is denoted $P_s$ and one particular permutation as $o \in P_s$. An extension $H'$ of $H$ onto SNP locus $s$ can be defined by appending some permutation $o \in P_s$ of alleles to $H$; $H' = H + o$. Note that it is possible for two distinct permutations to result in the same $H'$: $H + o = H + o'$. In these cases we do not include duplicates, as they are equivalent. Observe that if $H$ is empty, all allele permutations are the same as vector sets; we therefore include only one. For any $H'$, we can compute the probability of it being the correct haplotype (for the first $m+1$ SNPs) conditioning on $H$ being correct (for the first $m$ SNPs), as well as the semi-read data $SR(s)$ and error rate $\varepsilon$. We express this below:

$$P[H'|H, SR(s), \varepsilon] = \frac{P[SR(s)|H', H, \varepsilon] P[H'|H, \varepsilon]}{P[SR(s)|H, \varepsilon]}$$  \hspace{1cm} (4.2)

This computation is similar to those done above in *Equation 4.1*. The EXTEND algorithm is given below, which returns a list of all extensions $H'$ of $H$ that occur with probability above a certain threshold, $\rho$, given haplotype $H$.

**Branching**

Here we define branching a collection of haplotypes $\mathcal{H}$ with threshold $\rho$ to SNP $s$: BRANCH($\mathcal{H}, \rho, s$). We assume all $H \in \mathcal{H}$ phase the first $m \geq 0$ SNPs and that SNP $s$ is the $(m + 1)^{th}$ SNP. The act of branching $\mathcal{H}$ returns $\mathcal{H}'$: a list of all extensions generated by EXTEND with threshold $\rho$ for all $H$ in $\mathcal{H}$. To initialize BRANCH we EXTEND the empty vector set to an arbitrary permutation of the alleles of the first SNP, as all permutations are equivalent as vector sets.
Algorithm 1 EXTEND($H, \rho, s$): Extending a haplotype $H$ at SNP $s$ to all $H'$ that occur with probability $\geq \rho \in [0,1)$.

Input: $H, \rho, s$  
Output: $E'$  

```plaintext
E = []  
for $o \in P_s$ do  
    $H' = H + o$  
    if $H' \notin E$ then  
        if $P[H'|H, SR(s), \varepsilon] > \rho$ then  
            $E += H'$  
    return $E$
```

Algorithm 2 BRANCH($\mathcal{H}, \rho, s$): Branching haplotypes $\mathcal{H}$ at SNP $s$ with threshold $\rho \in [0,1)$.

Input: $\mathcal{H}, \rho, s$  
Output: $\mathcal{H}'$  

```plaintext
$\mathcal{H}' = []$  
for $H \in \mathcal{H}$ do  
    $E = \text{EXTEND}(H, \rho, s)$  
    for $H' \in E$ do  
        $\mathcal{H} += H'$  
    return $\mathcal{H}$
```

Pruning

For a collection of haplotypes $\mathcal{H}$ of SNPs $S' \subset S$, we can compute the relative likelihood of each haplotype conditioned on the sub-reads $R(S')$ and error rate $\varepsilon$; we write this as $RL[H|R(S'), \varepsilon]$. The same computation as performed in Equation 4.1 yields:

$$P[H|R(S'), \varepsilon] = \frac{P[R(S')|H, \varepsilon]P[H|\varepsilon]}{P[R(S')|\varepsilon]}.$$

Since $P[R(S')|\varepsilon]$ does not depend on $H$:

$$RL[H|R(S'), \varepsilon] = P[R(S')|H, \varepsilon]P[H|\varepsilon].$$

(4.3)

The goal of PRUNE($\mathcal{H}, \kappa, S'$) is to return a subset $\mathcal{H}' \subset \mathcal{H}$ containing only sufficiently probable haplotypes. It does so by computing the relative likelihood of the
most probable $H \in \mathcal{H}$: $\omega = \max_{H \in \mathcal{H}} \text{RL}[H|R(S'),\varepsilon]$ and adding $H \in \mathcal{H}$ to $\mathcal{H}'$ if
\[
\text{RL}[H|R(S'),\varepsilon] \geq \kappa \omega,
\]
where $\kappa$ is between 0 and 1. We note that that one can compute RL($H'$) from RL($H$) by only looking at the semi-reads $RS(s)$: we store the relative likelihood values for all $H \in \mathcal{H}$ and update them when branching to $\mathcal{H}'$; PRUNE is therefore no more costly than BRANCH.

Main algorithm

Here we give a high-level description of our overall haplotype assembly method using the EXTEND, BRANCH, and PRUNE algorithms. We generate high likelihood phases for the first $m$ SNPs, BRANCH those phases to include $s$ (the $(m+1)^{th}$ SNP), then PRUNE the resulting phases, and repeat for $m = m+1$. We begin with an arbitrary permutation of the first SNP, since all orderings result in the same vector set.

For the final step, we PRUNE with $\kappa = 1$, and therefore return only the maximally probable phases that we have found; if this set is of size greater than one, we choose a phasing from within it randomly. More generally, below we take $\rho$ and $\kappa$ to be vectors, as $\rho$ and $\kappa$ may depend on $m$, the size of $\mathcal{H}$ or other user-specified variables.

Algorithm 3 PRUNE($\mathcal{H}, \kappa, S'$): Pruning haplotypes $\mathcal{H}$ on $S'$ with factor $\kappa \in [0,1]$.

**Input:** $\mathcal{H}, \kappa, S'$

**Output:** $\mathcal{H}'$

$\mathcal{H}' = []$

$\omega = \max_{H \in \mathcal{H}} \text{RL}[H|R(S'),\varepsilon]$

for $H \in \mathcal{H}$ do

if $\text{RL}[H|R(S'),\varepsilon] \geq \kappa \omega$ then

$\mathcal{H}' += H$

end if

end for

return $\mathcal{H}$
Algorithm 4 HapTree($R, \hat{\rho}, \hat{\kappa}, S$): Assembling haplotype from reads $R$ with parameters $\hat{\rho}, \hat{\kappa}$.

**Input:** $R, \hat{\rho}, \hat{\kappa}, S$

**Output:** $\mathcal{H}$

\[
\mathcal{H} = []
\]

\[
S' = {}
\]

for $s \in [1, 2, ..., |S|]$ do

\[
S' += s
\]

\[
\mathcal{H} = \text{BRANCH}(\mathcal{H}, \hat{\rho}(s), s)
\]

\[
\mathcal{H} = \text{PRUNE}(\mathcal{H}, \hat{\kappa}(s), S')
\]

return $\mathcal{H}$


4.3 Results

4.3.1 Scoring and evaluation

Determining the quality of a phasing solution depends on whether the true phase is known. When no such information is available, the Minimum Error Correction (MEC) score [103] is a widely used scoring function to measure the quality of phasing solutions. The MEC score is defined as the minimum (amongst chromosomes) number of mismatches between a phase $H$ and the read set $R$. A number of existing programs, including HapCut [97], find phasing solutions by optimizing the MEC score in diploid cases. For higher ploidy the MEC score can no longer be reliably used because unlike in the diploid case, the phase of any one chromosome does not determine the phases of the others. Moreover, the MEC score does not distinguish between two separate phases of a pair of SNP loci with different non-zero counts of $(0,0), (0,1), (1,0), (1,1)$ in their vector sets. Finally, unlike in the diploid case, a phase of a pair of SNP loci containing a set of parallel alleles does not prevent it from containing a set of switched alleles as well. To demonstrate these issues, consider two possible vector sets corresponding to phases of a pair of triploid SNPs both with genotype 2: $a: ((0,0), (0,0), (1,1))$ and $b: ((0,0), (0,1), (1,0))$. If the read data is $((0,0), (0,0), (0,0))$, it is clear from a probabilistic standpoint that phase $a$ is a better fit, but both $a$ and $b$ have equal MEC scores. This effect is exaggerated as $k$ increases.
When a true phase is available, there are ways to evaluate how accurate any predicted phase is. A widely used measure in diploid phasing is switch error, which is calculated as the number of positions where the two chromosomes of a proposed phase must be switched in order to agree with the true phase. For polyploid phasing, we generalize switch error to vector error. In higher ploidy cases, at any SNP locus, it is possible for 0 chromosomes of a proposed phase to require switches or anywhere from 2 to $k$ chromosomes to require switches, in order for a proposed phase to agree with the true phase. We do not wish to penalize a solution where only two vectors must be switched at a given position as much as we would penalize a solution in which all vectors must be switched. The vector error of a proposed phase with respect to the true phase is defined by the minimum number of segments on all chromosomes for which a switch must occur; for the diploid case this score is exactly twice the switch error. One may also think of the vector error as the minimum number of segments a proposed phase and the true phase have in common, less the ploidy. Even for triploid genomes, the vector error is more discriminative and subtle than switch error. Consider the following example that shows a vector error in a sample tetraploid genome; the true phase is on the left and examples with two, three, and four vector errors are on the right:

<table>
<thead>
<tr>
<th>True Phase</th>
<th>(i): 2 Vector Errors</th>
<th>(ii): 3 Vector Errors</th>
<th>(iii): 4 Vector Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0 0 1</td>
<td>1 1 1 0 0 0 1</td>
<td>1 1 1 1 0 1</td>
<td>1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1 0 0 1 0</td>
<td>1 0 1 1 1 0 1</td>
<td>1 0 1 0 0 1</td>
<td>1 0 1 1 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1 1 0 1</td>
<td>0 0 0 0 1 0</td>
<td>0 0 0 0 1 0</td>
<td>0 0 0 0 1 1</td>
</tr>
</tbody>
</table>

In this case, phase (i) is a more accurate phase than (ii), and phase (ii) more accurate than phase (iii). The segments are broken up by row and color: phase (i) having five segments, phase (ii) having six, and phase (iii) having seven. Note that there may be several ways to break a vector set into a minimal number of segments; phase (ii) is such an example. Finally, we remark that vector error can be computed efficiently in time $O(kn^2)$, where $k$ is the ploidy and $n$ the block size.
4.3.2 Results for simulated polyploid data

Simulating polyploid genomes

To simulate a genome, we fix a ploidy \((k)\) and the number of SNPs \((n)\). We determine the positions for the SNPs by randomly generating the distance between each pair of adjacent SNPs. We do so using a geometric random variable with parameter \(p\) (SNP density); this choice is equivalent to assuming that any position is a SNP independently with probability \(p\). For phasing purposes, once one has generated the reads, the exact genomic positions are no longer relevant; they were only needed to simulate more accurate read data. We therefore refer to SNPs by their position amongst the SNPs, not their position in the genome. For each SNP, we randomly generate its haplotype, assuming for each chromosome, that the alternative and reference alleles are equally likely; if we generate a homozygous SNP, we try again. This procedure results in the likelihood of genotype \(g(s) \in \{1, \ldots, k - 1\}\) equal to \(\binom{k}{g(s)}/(2^k - 2)\), and all orderings \(o \in P_s\) being equally likely. For the simulations discussed we use this model. Note, however, that HapTree is not dependent on this model.

Simulating read data

To generate a paired-end read, we uniformly choose a starting point on the genome (we make sure the genome starts sufficiently before the first SNP and ends at the last). We fix the read-end length \(\text{read\_len}\) to be 150. The fragment length \(\text{frag\_len}\) is normally distributed with a mean of 550 and standard deviation of 30, but with min and max lengths of 500 and 600 respectively. The insert length is determined by the fragment length and read-end length, that is, insert length = fragment length - 2 \times read length. Once we know the start and fragment length, we must choose from which chromosome to read; we do so uniformly from the \(k\) chromosomes. Finally, we add uniform error to the read; we choose a rate of .02, based on the reported error rate of Illumina sequencing technologies. For every SNP that the read covers, independently with probability \(\epsilon\) we flip the allele to any other allele; two-thirds of the time when we have this error, we can see that the allele present is neither the reference nor the
alternative, and therefore we delete it. Hence, conditional on seeing a SNP in a read, it is incorrect with probability $\varepsilon = \frac{e^{-\frac{C}{1 - 3\varepsilon}}}{1 - 3\varepsilon}$ and correct with probability $1 - \varepsilon$.

For any genome, to generate a read set with $Cx$ coverage we need each base pair to be on average covered by $C$ reads. To determine the number of reads to generate, we must know the length of the genome and the read length. The expected length of the genome is $\frac{n}{p}$ for SNP density $p$, and the read length is 150bp for each end (of which there are two); therefore we simulate $\frac{Cn}{300p}$ reads for $C \times$ coverage. Note that many of these reads will see only zero or one SNP(s), thus for $C \times$ coverage the number of useful reads for any SNP will be less than $C$.

Relative likelihood (RL) objective function vs. MEC score for polyploid genomes

We assessed the effectiveness of RL score by comparison to MEC score on simulated data. To do so, we simulated reads with error rate 0.02 from a pair of phased $k$-ploid SNP loci for different coverages (5x, 10x, 20x, 100x) and for $k \in \{2, ..., 10\}$. All possible phases were exhaustively enumerated, and phases of the maximal relative likelihood (RL) and phases of the minimal MEC score chosen. We computed the proportion of perfectly phased SNP pairs in both cases (perfect solution rate). Even with two SNP loci, RL significantly outperforms MEC for all $k \geq 3$ (Figure 4-2A). It is also worth noting that MEC (in comparison to RL) deteriorates more seriously in accuracy as ploidy ($k$) increases (Figure 4-2A). In addition, we also compared the vector error rate in both cases; for a pair of SNPs, this rate is the number of vectors from the proposed solution that cannot be matched with vectors from the true solution (Figure 4-2B).

The results demonstrate that the higher the ploidy, the better the relative likelihood (RL) score performs in comparison to MEC score for phasing a pair of SNPs (Figure 4-2). In fact, in simulations where $k \geq 8$, RL with 5x the coverage already outperforms MEC with 100x coverage. For the same coverage, RL always outperforms MEC for $k \geq 3$, and they are equivalent in the diploid case ($k = 2$).
Figure 4-2: Proportion of perfectly phased SNP pairs and vector error rate for RL (solid line) and MEC (dashed line) optimization in 10000 trials over 5x, 10x, 20x and 100x coverage.

Comparisons of HapTree and HapCompass

To evaluate the phasing capabilities of HapTree, we compare it with HapCompass [96], to our knowledge the only other existing program that directly addresses polyploid haplotype assembly, over multiple depth coverage values and component sizes for triploid and tetraploid simulated genomes. We simulated triploid and tetraploid genomes with different block lengths (10, 20 or 40 SNP loci), different coverages (5x, 10x, 20x and 40x), SNP positions, and SNP densities. Throughout the simulations for both the triploid and tetraploid cases, our EXTEND module is run with threshold $\rho = .01$ and PRUNE primarily with threshold $\kappa = .001$. When the current number of haplotype options generated is above 1000, we prune more aggressively with $\kappa = .01$ and when above 5000, with $\kappa = .05$. These parameters are chosen to ensure the efficiency of HapTree by only keep a tractable collection of promising solutions in each step. We also simulate a read set with uniform error rate and size dependent on coverage.

For the triploid case, we observed that HapTree finds a perfect solution at a rate independent of the number of SNPs used in the simulation; in contrast, HapCompass declines in performance the larger the block size (Figure 4-3). While both HapTree and HapCompass improve steadily the higher the coverage, in every case HapTree significantly outperforms HapCompass; the least significant improvement of 63% occurs in the case of 10 SNP loci and 10x coverage, whereas the most significant improvement occurs in the case of 40 SNP loci and 40x coverage. For both vector error rate and likelihood of perfect solution, we find that HapTree substantially outperforms
Figure 4-3: HapTree (solid lines) and HapCompass (dashed lines) on simulated triploid genomes: Likelihood of Perfect Solution and Vector Error Rates, 1000 Trials, Block lengths: 10, 20, and 40.

Figure 4-4: HapTree (solid line) and HapCompass (dashed line) on simulated tetraploid genomes: Likelihood of Perfect Solution and Vector Error Rates, 1000 Trials, Block length: 10.

HapCompass.

For tetraploid simulations, HapTree significantly outperforms HapCompass with block length of 10 SNP loci (Figure 4-4). For larger block lengths HapCompass arrives at the perfect solution at a rate of less than 1%; HapTree however does so at a rate between 40% and 70% depending on block size and coverage at least 20×.

We varied the allele error rates (.001, .02, .05, and .1) and observed decreases in accuracy that vary approximately linearly with the (uniform) allele error rates. The allele error rate is the likelihood of the sequencing technology to report the incorrect allele for one position in one read. We ran 10000 trials for simulated triploid genomes of block size 10, with coverages 10×, 20×, and 40×.

Run-time evaluation. Not only does HapTree outperform HapCompass on phasing quality, it is also significantly faster, especially for longer block length. The median runtimes for block length 10 and 10× coverage were 0.00702, and 0.633 seconds for HapTree and HapCompass, respectively; for block length of 40 and 40× coverage, they were 0.0279, and 13.099 seconds, respectively.
Figure 4-5: HapTree performance over varied error rates (.001, .02, .05, .1) and coverages (10×, 20×, 40×) on simulated triploid genomes: Likelihood of Perfect Solution and Vector Error Rates, 10000 Trials, Block length: 10.

4.3.3 Results for real diploid data

As seen in the results of Geraci et al. [105], there is no perfect solution for diploid phasing. HapCUT is one of the methods reported that consistently performs best or close-to-best for a variety of experiments. For a proof of concept of how HapTree would perform on real data, we ran HapTree and HapCUT using 454 and Illumina sequencing data of the well-studied NA12878 genome (1000 Genomes Project Phase 1) [83], and compared MEC scores as well as switch errors to a trio phasing annotation accepted as ground truth; we present these results in Table 4.1. The trio phasing annotation represents a high quality diplotype of NA12878 for all SNP sites where either parent (NA12891 or NA12892) is homozygous [83]. Note that we computed the number of switch errors within connected SNP components only, against SNPs whose phase has been determined by the trio-based phasing; we then sum over components. In this case, HapTree was run with a uniform error rate of .02, an EXTEND threshold .001, and primarily with a PRUNE threshold of .001. We begin to prune more aggressively when we have at least 100, 500, or 1000 possible haplotypes with thresholds of (.01, 05, .1) respectively. For the vector set prior, from examining the read data, we ran HapTree with parallel bias \( p = .8 \).

We found that HapTree and HapCUT perform almost identically in MEC scores, with HapCUT having marginally smaller scores for both 454 and Illumina datasets. It is worth noting that HapCUT optimizes MEC score, and MEC score measures only the consistency between a phasing solution and read data, not with the true phase.

Notably, when comparing to the ground-truth phase as determined by trio-based
Table 4.1: Results of switch error and MEC score for HapTree and HapCUT of whole-genome phasing using 454 and Illumina sequencing datasets. A lower value is better for each scoring metric, where MEC score measures the consistency of phase decisions made given the read dataset, and switch error measures the number of phase mistakes made compared to a golden truth dataset.

<table>
<thead>
<tr>
<th></th>
<th>454</th>
<th>Illumina</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEC</td>
<td>Switch Error</td>
</tr>
<tr>
<td>HapTree</td>
<td>32818</td>
<td>2978</td>
</tr>
<tr>
<td>HapCUT</td>
<td>32781</td>
<td>3192</td>
</tr>
</tbody>
</table>

phasing, we found HapTree significantly outperforms HapCut in terms of switch error rate for the phasing experiments on the NA12878 genome for 454 and Illumina datasets. Although our method is not primarily designed for phasing diploid genomes, it is still able to achieve better phasing results, when compared to the state-of-the-art diploid method. Again, the results on real-world read datasets showed the superiority of our likelihood function over MEC score for NGS-based phasing.

For full diploid genome phasing, HapTree ran in approximately 3 minutes, while HapCut took between 8-14 minutes depending on the number of iterations.
Chapter 5

Haplotype Reconstruction of the Transcriptome

5.1 Overview

As more sequencing data becomes available [9], we seek to design efficient algorithms to obtain accurate and comprehensive phase information directly from transcriptomic, as well as the commonly-used genomic, NGS read data. Transcriptome sequencing data differs from genomic read data in that genes often have differential haplotypic expression [106]. We wish to leverage this asymmetry to increase the number of SNPs of an individual that can be phased.

Methods have been proposed for the computational identification of an individual’s diplotype using pedigree (e.g. trio-based phasing) [101, 89], population structure of variants (e.g. phasing by linkage disequilibrium) [94, 91, 107, 108] and more recently by identity-by-descent in unrelated individuals [95, 96], as well as various types of sequencing read datasets: DNA-seq [97, 100, 98, 99, 109], RNA-seq [35]; proximity-ligation [110]; and haploid sperm sequencing [111]. Population-based, IBD and pedigree-based methods require data from a group of individuals to perform phasing of a individual. For solving the single-individual haplotype reconstruction problem, using RNA-seq or DNA-seq data are the most viable approaches as they are widely-available and inexpensive.
Long-range diplotyping is important because it gives more statistical power for downstream analyses [110]. Compared to DNA-seq, RNA-seq allows for longer-range phasing due to RNA splicing in the transcriptome. To date, approaches that utilize RNA-seq data for phasing (e.g., [35]) can only make use of reads covering 2 or more heterozygous SNPs, as they repurpose existing genome phasing approaches which are based on sequence contiguity. However, only 10% of reads that overlap a heterozygous SNP fall into this category (Table 5.1). Thus, current methods are discarding 90% of potentially useful information. Though these reads do not overlap multiple SNPs, whereas reads conventionally used for phasing do, they still provide insight into differential haplotypic expression within genes. An advantage of using reads covering only a single SNP is that phasing is not limited by the length of the read or fragment, nor the transcriptomic or genomic distance between SNPs (Figure 5-1).

In this chapter, we present a novel method for solving the haplotype reconstruction problem using differential allele-specific expression (DASE) information within RNA-seq data. We follow the intuition that DASE in the transcriptome can be exploited to improve phasing power because SNP alleles within maternal and paternal haplotypes of a gene are present in the read data at (different) frequencies corresponding to the differential haplotypic expression [106]. To solve this haplotype reconstruction problem, we introduce a new maximum-likelihood formulation, generalizing that from HapTree [109], which considers DASE and is thus able to exploit reads covering only one SNP. This formulation results in an integrative algorithm, HapTree-X; which determines a haplotype of maximal likelihood based on both RNA-seq and DNA-seq read data. Our method does not require fragments to cover at least two SNPs and therefore for the first time can leverage the large number of RNA-seq fragments that cover only one SNP to produce more accurate phasing, as well as both novel and larger blocks of phased SNPs.

Reported RNA-seq phasing results using HapTree-X for a well annotated human lymphoblastoid cell line (GM12878) provide strong evidence for long-distance haplotype phasing capability of paired-end RNA-seq read alignments as well as the use of differential allele-specific expression as a practical haplotype reconstruction tool.
Used jointly with genome reads in genotyping studies, RNA-seq reads can provide long distance scaffolds in order to be used for extending and merging haplotypes inferred from genome reads as well as introducing new long-distance phasing instances not possible to attain using short genome sequencing reads. We observe that compared to a state-of-the-art sequence-based haplotype reconstruction method, HapCut [97], HapTree-X increases the total number of SNPs phased along with the sizes of phased haplotype blocks with improved accuracy, leveraging RNA-seq reads that only cover a single heterozygous-SNP in the transcriptome.

5.2 Methods

5.2.1 Definitions and notation

In this section we provide technical notation and discuss several basic assumptions we make.

The goal of phasing is to recover the unknown haplotypes (haploid genotypes), \( H = (H_0, H_1) \), which contain the sequence of variant alleles inherited from each parent of the individual. As homozygous SNPs are irrelevant for phasing, we restrict ourselves to heterozygous SNPs (from now on referred to simply as a ‘SNP’) and we denote the set of these SNPs as \( S \). We assume these SNPs to be biallelic, and because of these restrictions, \( H_0 \) and \( H_1 \) are complements. Let \( H[s] = (H_0[s], H_1[s]) \) denote the alleles present at \( s \), for \( s \in S \).

We denote the sequence of observed nucleotides of a fragment simply as a "read" (independent from single/paired-end reads). We assume each read is mapped accurately and uniquely to the reference genome, and moreover that each read is sampled independently. The set of all reads is denoted as \( R \). Given a set of SNP loci \( S \), we define a read \( r \in R \) as a vector with entries \( r[s] \in \{0, 1, -\} \), for \( s \in S \), where a 0 denotes the reference allele, a 1 the alternative allele, and - that the read does not overlap \( s \) or that it contains false allele information at \( s \). We say a read \( r \in R \) contains a SNP \( s \) if \( r[s] \neq - \) and we let size of a read \( r \), \( |r| \), refer to the number of SNPs it contains.
Figure 5-1: A toy example demonstrating the haplotype phasing capabilities of and differences between single-individual haplotype reconstruction methods using genome sequencing (DNA-seq) reads (a), transcriptome sequencing (RNA-seq) reads (b), and differential allele-specific expression (DASE) information that can be inferred from RNA-seq data (c). Green and orange blocks respectively represent reference genome and the transcriptome sequence, which contains only the exons in a gene separated by introns. Positions marked in red denote heterozygous-SNP loci. Paired-end sequencing reads are of length 2x4bp and have 3-4 bp insert lengths; reference alleles overlapping SNP loci are marked with red and alternative alleles are marked with blue. (a) Phasing using DNA-seq reads can be performed by looking at reads that overlap multiple heterozygous-SNP loci and observing the alleles that are connected through reads. Phasing distance is limited by maximum fragment length (12bp in the example). Multiple SNP loci can be chained together for phasing, but the probability of a switch error increases with the length of the chain. (b) Though limited to only the SNPs within the transcriptome, RNA-seq reads have longer distance phasing capability than DNA-seq reads due to long introns in the genome that are spliced-out in the sequenced transcript fragments. RNA-seq reads also provide higher accuracy phasing of SNPs within the transcriptome compared to DNA-seq, since DNA-seq phasing needs to chain through intron SNPs to connect the exons. (c) Differential allele-specific expression (DASE) at transcriptomic SNP loci is available within RNA-seq datasets in the form of allele-specific coverage ratios. For genes that display differential haplotypic expression (DHE), the majority of alleles can be phased together to obtain a single haplotype block for the entire gene. Depending on the DHE and depth-coverage, DASE-based phasing can perform accurate haplotype reconstruction, independent of gene/exon lengths, without requiring paired-end or long reads.
Figure 5-1: (cont.) (d) Phasing capabilities of DNA-seq, RNA-seq and DASE based phasing methods are demonstrated on the given toy example. The genome sequencing based approach is only able to provide haplotype blocks for the exons close together. The RNA-seq read based approach is able to reconstruct a longer haplotype block, phasing through the introns as well, but failing to phase far apart SNPs within the first exon. Whereas DASE-based phasing is able to reconstruct the complete gene haplotype by leveraging differential expression at SNP loci.

For each read $r$ and for each SNP locus $s$, we assume a probability of opposite allele (reference if the true allele is the alternative, and vice versa) information $r[s]$ equal to $\varepsilon_{r,s}$ and represent these error probabilities as a matrix $\varepsilon$. We assume these errors to be independent from one another.

In genomic read data, all $r \in R$ are equally likely to be sampled from the maternal or paternal chromosomes. In RNA-seq data however, this may not always be the case. We define the differential haplotypic expression (DHE) to represent the underlying expression bias between the maternal and paternal chromosomes of a particular gene. Throughout, we will refer to the probability of sampling from the higher frequency haplotype of a gene as $\beta$. We assume two genes $g, g'$ have independent expression biases $\beta, \beta'$. Differential allele-specific expression (DASE) we define as the observed bias in the alleles at a particular SNP locus present in $R$. We define concordant expression as when the DASE of a SNP agrees with the DHE of the gene to which the SNP belongs; that is when the majority allele (allele occurring with higher frequency) occurring within the reads at a particular SNP locus is in agreement with the expected majority allele as determined by the DHE.

To perform phasing using the sequence contiguity within reads (contig-based phasing), upon the set of SNP loci $S$ and read set $R$, we define a 'read graph' such that there is a vertex for each SNP locus $s \in S$ and an edge between any two vertices $s, s'$ if there exists some read $r$ containing both $s$ and $s'$. These connected components correspond to the haplotype blocks to be phased.

To phase using differential expression (DASE-based phasing), we assume the existence of some gene model $G$ that specifies the genes (and their exons) within the genome. For each $g \in G$, we assume that the haplotypes $(H_0, H_1)$ restricted to $g$ are
expressed at rates $\beta_0, \beta_1$ respectively due to DHE. The phasing blocks correspond to the SNPs in genes $g \in G$, though we will see that some SNPs are not phased due to insufficient probability of concordant expression. Two distinct genes $g, g'$ may not be DASE-phased due to lack of correlation between their expression biases $\beta, \beta'$. In the remainder of this chapter, when DASE-phasing a particular gene, by $H$ we mean the gene haplotype, that is $H$ restricted to the SNPs within $g$.

The blocks which are able to be phased by HapTree-X integrating both contig and DASE-based phasing are defined as the connected components of a Joint Read Graph. In the Joint Read Graph, each vertex corresponds to a SNP phased by either method, and there is an edge between any two vertices (SNPs) $s, s'$ if there exists some block that was phased by either method containing both $s, s'$.

Building on the definitions above, we describe the mathematical underpinnings of the haplotype reconstruction problem that assume the existence of DHE. We include an overview of our algorithm in subsection 5.2.5.

5.2.2 Likelihood of a phase

We formulate the haplotype reconstruction problem as identifying the most likely phase(s) of set of SNPs $S$, given the read data $R$, and sequencing error rates $\varepsilon$ \footnote{Note that $\varepsilon$ is the probability of opposite false allele as opposed to the physical sequencing error rate per base, which was denoted as $\epsilon$ in Chapter 4. We assume the relation between the two to be $\varepsilon = \frac{\epsilon}{1 - \frac{\epsilon}{2}}$.}. Furthermore, suppose we knew for each read $r$, the likelihood that $r$ was sampled from $H_i$ (denote this as $\beta_i$); we represent these probabilities as a matrix $B$. While $B$ is not given to us, we may estimate $B$ from $R$. We derive a likelihood equation for $H$, conditional on $R, B$ and $\varepsilon$.

Given a haplotype $H$, reads $R$, error rates $\varepsilon$, and $B$, the likelihood of $H$ being the true phase is

$$P[H|R, B, \varepsilon] = \frac{P[R|H, B, \varepsilon] P[H|B, \varepsilon]}{P[R|B, \varepsilon]}. \quad (5.1)$$

Since $P[R|B, \varepsilon]$ does not depend on $H$, we may define a relative likelihood measure, RL. Note that $P[H|B, \varepsilon] = P[H]$ as the priors on the haplotypes are independent of...
the errors in $R$, and of $B$.

$$RL[H|R, B, \varepsilon] = P[R|H, B, \varepsilon] P[H].$$  \hspace{1cm} (5.2)

For the prior $P[H]$, we assume a potential parallel bias, $\rho \geq .5$, (the prior probability of adjacent SNPs being phased in parallel as opposed to switched) which results in a distribution on $H$ such that adjacent SNPs are independently believed to be phased in parallel $\binom{00}{11}$ with probability \(\rho\) and switched $\binom{01}{10}$ with probability $1 - \rho$. When $\rho = .5$ we have the uniform distribution on $H$. The general prior distribution on $H$ in terms of $\rho$ is

$$P[H] = \rho^P(H)(1 - \rho)^S(H)$$  \hspace{1cm} (5.3)

where $P(H)$ and $S(H)$ denote the number of adjacent SNPs that are parallel and switched in $H$, respectively. Given the above model, as each $r \in R$ independent, we may expand $P[R|H, B, \varepsilon]$ as a product:

$$P[R|H, B, \varepsilon] = \prod_{r \in R} P[r|H, B, \varepsilon]$$  \hspace{1cm} (5.4)

In the setting of RNA-seq, reads are not sampled uniformly across homologous chromosomes, but rather according to the DHE (expression bias) of the gene from which they are transcribed. We see in (5.5) how this asymmetry allows us to incorporate reads which contain only one SNP. Let $A(r, H_i), D(r, H_i)$ denote the SNP loci where $r$ and $H_i$ agree and disagree respectively, then

$$P[r|H_i, [], \varepsilon] = \sum_{s \in [0,1]} \left( \beta_i^s \prod_{s \in A(r, H_i)} (1 - \varepsilon_{r,s}) \prod_{s \in D(r, H_i)} \varepsilon_{r,s} \right).$$  \hspace{1cm} (5.5)

When there is uniform expression $\beta_0^r = \beta_1^r$ (no bias) and if $|r| = 1$, then $P[r|H, B, \varepsilon]$ is constant across all $H$. This is not the case when the expression bias is present however, and therefore reads covering only one SNP affect the likelihood of $H$.

If we knew the matrix $B$, we could apply HapTree [109] to find $H$ of maximal likelihood; the matrix $B$, however, is unknown. Throughout this chapter we provide
methods for determining a maximum likelihood $B$, and for which reads $r$ we are sufficiently confident there this is in fact non-uniform expression, that is $\beta_0^r \neq \beta_1^r$. Moreover, we determine for which SNPs $s \in S$ (contained only by reads of size one), we have sufficient coverage and expression bias to determine with high accuracy the phase $H[s]$.

5.2.3 Maximum likelihood differential haplotypic expression

For a fixed gene $g$, containing SNPs $S_g$, the corresponding reads $R_g$ have expression biases $\beta_0^r, \beta_1^r$ which are constant across $r \in R_g$. Let $\beta = \beta_0^r$ refer to this common expression; we wish to determine the maximum likelihood underlying expression bias $\beta$ of $g$ responsible for producing $R_g$. To do so, we formulate a Hidden Markov Model (HMM) and use the forward algorithm to compute relative likelihoods of $R$ given $\beta, \varepsilon$.

To achieve the conditional independence required in a HMM, we define $R'_g$, a modification of $R_g$, containing only reads of size one, so that $R'_{g,s}$ (the reads $r \in R'_g$ which cover $s$) are independent from $R'_{g,s'} (\forall s \neq s' \in S_g)$. We restrict each $r \in R_g$ to a uniformly random SNP $s$, and include this restricted read of size one $(r|s)$ in $R'_g$ (we note that if $|r| = 1$, then $r = r|s$, by definition.) Therefore, $R'_{g,s}$ and $R'_{g,s'}$ are independent as all $r \in R'_g$ are of size one.

Our goal is the determine the maximum likelihood $\beta$, given $R'_g$. We assume a uniform prior on $\beta$, and therefore $P[\beta | R'_g, \varepsilon]$ is proportional to $P[R'_g | \beta, \varepsilon]$ (immediate from Bayes theorem). We may theoretically compute $P[R'_g | \beta, \varepsilon]$ by conditioning $H$ (which is independent from $\beta, \varepsilon$)

$$P[R'_g | \beta, \varepsilon] = \sum_H P[R'_g | H, \beta, \varepsilon] P[H]$$

and expand $P[R'_g | H, \beta, \varepsilon]$ as a product over $r \in R'_g$ as in (5.4) and (5.5). This method, however, requires enumerating all $H$; since $|H| = 2^{|S_g|}$ we seek different approach. Indeed, we translate this process into the framework of a Hidden Markov Model, apply the forward algorithm to compute $f(\beta) := P[R'_g | \beta, \varepsilon]$ exactly for any $\beta$, and since $f$ has a unique local maxima for $\beta \in [0.5, 1]$, we can apply Newton-Rhapson.
method to determine $\beta$ of maximum likelihood.

To set this problem in the framework of a Hidden Markov Model, we let the haplotypes $H$ correspond to the hidden states, $R'_g$ to the observations, and let the time evolution be the ordering of the SNPs $S_g$. The observation at time $s$ in this context is $R'_{g,s}$, the reads covering SNP $s$. The emission distributions are as follows:

$$P[R'_{g,s}|H[s],\beta,\varepsilon] = \prod_{r \in R'_{g,s}} P[r|H[s],\beta,\varepsilon]$$

$$P[r|H[s],\beta,\varepsilon] = \begin{cases} 
\beta_0(1 - \varepsilon_{r,s}) + (1 - \beta_0)\varepsilon_{r,s} & \text{if } r[s] = H_0[s] \\
\beta_1(1 - \varepsilon_{r,s}) + (1 - \beta_1)\varepsilon_{r,s} & \text{if } r[s] = H_1[s]
\end{cases} \quad (5.6)$$

where $H[s]$ is $H$ restricted to $s$.

To determine the hidden state transition probabilities, recall our prior on $H$ in (5.3). We may equivalently model this distribution $H$ as a Markov chain, with transition probabilities:

$$P[H[s_{i+1}]|H[s_i]] = \begin{cases} 
\rho & \text{if } H_0[s_i] = H_0[s_{i+1}] \\
1 - \rho & \text{if } H_0[s_i] \neq H_0[s_{i+1}]
\end{cases}$$

These emission probabilities and hidden state transition probabilities are all that are needed to apply the forward algorithm and determine the $\beta$ of maximum likelihood.

### 5.2.4 Likelihood of concordant expression

**A solution of maximum likelihood**

In this subsection, we see the role played by concordant expression, and motivate its use as a probabilistic measure for determining which SNPs we believe we may phase with high accuracy.

We derive $H^+$, a haplotype solution of a gene $g$, of maximum likelihood given $R'_g$, $\beta$ and $\varepsilon$ and conditions CND1, CND2, and CND3. Let $C'_s$ denote the number of reads $r \in R'_{g,s}$ such that $r[s] = v$ where $v \in \{0, 1\}$. Provided error rates are constant
(CND1) (say $\epsilon$) and $\epsilon < .5$ (CND2), and assuming a uniform prior distribution ($\rho = .5$) (CND3), we can show a solution of maximum likelihood is $H^+ = (H_0^+, H_1^+)$, where $H_0^+[s] = \nu$ such that $C_s^\nu \geq C_s^{1-\nu}$. In words, $H_0^+$ and $H_1^+$ contain the alleles that are expressed the majority and minority of the time (respectively) at each SNP locus; given sufficient expression bias and coverage, intuitively, $H^+$ ought to correctly recover the true haplotypes. It is easy to show that CND1 and CND3 can be removed if one is willing to specify a minimum coverage; we do not show this here. Intuitively, CND2 must not be removed.

To prove $H^+$ is of maximal likelihood, we introduce the terms ‘concordant expression’ and ‘discordant expression’. We say $R$ and $H$ have concordant expression at $s$ if $C_s^{H_0[s]} > C_s^{H_1[s]}$, discordant expression if $C_s^{H_0[s]} < C_s^{H_1[s]}$, and equal expression otherwise. In words, since we assume $\beta_0 > \beta_1$, we expect to see the allele $H_0[s]$ expressed more than the allele $H_1[s]$ in $R_{g,s}$ (concordant expression).

We may now equivalently define $H^+$ as a solution which assumes concordant or equal expression at every SNP $s$. Because we assume uniform priors, $P[H|R_g', \beta, \epsilon]$ is proportional $P[R_g'|H, \beta, \epsilon]$ (see (5.1)), and since each read is of size one, we can factor across $S_g$ in the following way:

$$P[R|H, \beta, \epsilon] = \prod_{s \in S_g} P[R_{g,s}|H[s], \beta, \epsilon]$$

Therefore, to show $H^+$ is of maximal likelihood, it only remains to show that concordant expression is at least as likely as discordant expression, as intuition suggests. Let $\gamma_i = \beta_i(1 - \epsilon) + (1 - \beta_i)\epsilon$, then as in (5.6) we may deduce

$$P[R_{g,s}|H[s], \beta, \epsilon] = \prod_{i \in \{0,1\}} \gamma_i^{C_s^{H_i[s]}}$$

Let $H^- = (H_1^+, H_0^+)$, the opposite of $H^+$. We can now compare the likelihood of concordant (or equal) expression at $s$ ($H^+[s]$) with that of discordant (or equal)
expression at \( s \) \((H^{-}[s])\). For ease of notation, let \( v_i = H_i^+[s] \) and \( w_i = H_i^-[s] \).

\[
\frac{\Pr[R_{g,s}|H^+[s],\beta,\epsilon]}{\Pr[R_{g,s}|H^-[s],\beta,\epsilon]} = \prod_{i \in \{0,1\}} \frac{\gamma_i^{C_i^+}}{\gamma_i^{C_i^-}} = \frac{\gamma_0^{C_0^0-C_0^0}}{\gamma_1^{C_0^+}} \geq 1 \tag{5.7}
\]

The rightmost equality results from the fact that \( H_i^+ = H_i^- \), and hence \( v_i = w_{1-i} \).

Since \( \epsilon < .5 \), we have \( \gamma_0 > \gamma_1 \); \( C_0^0 - C_0^+ \geq 0 \) by the definition of \( H^+ \), which proves the inequality.

**Computing likelihood of concordant expression**

We just showed that under mild conditions, the solution of maximal likelihood is, intuitively, that which has concordant expression at each SNP locus \( s \). Therefore, to determine which SNPs we believe we can phase with high accuracy, we measure the probability of concordant expression at that SNP, and only phase when that probability is sufficiently high.

These probability of concordant expression can be immediately derived from (5.7). We assume a uniform error rate of \( \epsilon \) for ease of notation, though is not required. Let \( CE(R_{g,s}, H[s]) \) denote the event of concordant expression at \( s \), then

\[
\Pr[CE(R_{g,s}, H[s])|\beta,\epsilon] = \frac{\Pr[R_{g,s}|H^+[s],\beta,\epsilon]}{\Pr[R_{g,s}|H^+[s],\beta,\epsilon] + \Pr[R_{g,s}|H^-[s],\beta,\epsilon]} = \frac{1}{1 + \left( \frac{\gamma_1}{\gamma_0} \right)^{|C_0^0-C_0^+|}} \tag{5.8}
\]

Furthermore, given \( N \) reads, an expression bias \( \beta \), and a constant error rate \( \epsilon \), we compute likelihood of concordant expression using the standard binomial distribution \( B(N, \gamma_0) \) by equating ‘successes’ in the binomial model to observations of the majority allele, expressed with bias \( \gamma_0 \) (recall \( \gamma_i \) takes errors into account):

\[
\Pr[CE|N,\beta,\epsilon] = \sum_{i=\lceil N/2 \rceil}^{N} \binom{N}{i} \gamma_0^i \gamma_1^{N-i} \geq 1 - e^{-N \frac{1}{2\gamma_0} (\gamma_0 - \frac{1}{2})^2} \tag{5.9}
\]

To obtain the bound on the right hand side, apply the Chernoff bound \( \Pr[X < (1 - \lambda)\mu] \leq e^{-\frac{\lambda^2}{2}} \) where \( X \) corresponds to the number of ‘successes’ and \( \mu = E[X] = N \beta \).
This bound shows that the probability of concordant expression increases exponentially with the coverage ($N$).

We remark for large $N$, the Binomial Distribution $B(n, \beta)$ converges to the normal distribution $\mathcal{N}(N\beta, N\beta(1 - \beta))$, and therefore this probability can always be easily computed. See Figures 5-2 and 5-3 for a sense of these likelihoods.

### 5.2.5 HapTree-X framework

HapTree-X is a novel Bayesian haplotype reconstruction framework tailored to RNA-seq read datasets that employs contig-based and DASE-based haplotype phasing within a joint probabilistic model.

HapTree-X outputs phased haplotype blocks, given an input of RNA-seq (and optionally in addition DNA-seq) read alignment files (bam/sam), a VCF file containing the individual’s genotype, and a gene model which specifies the genes (and their exons) within the genome.

The HapTree-X pipeline is initiated by determining which genes are expressed using the gene model and RNA-seq data. For each of these genes, a maximum likelihood expression bias (DHE) is computed. Furthermore, we determine which SNPs within those genes have high likelihood of concordant expression; we phase only those SNPs.

For reads containing only such SNPs, we assign to them the computed expression bias of the gene they cover; for all other reads, we assign a non-biased expression. Finally, applying a generalized version of HapTree [109], we determine a haplotype of maximal likelihood which depends on the DASE present in the RNA-seq data, as well as the sequence contiguity information within the reads.

HapTree-X is available at [http://groups.csail.mit.edu/cb/haptreex/](http://groups.csail.mit.edu/cb/haptreex/).
5.3 Results

5.3.1 Theoretical performance

We demonstrated earlier that the differential haplotypic expression level of a gene, $\beta$, and its coverage determine likelihood of concordant expression. We show this relationship below for varying $\beta$ and levels of coverage. While these functions are derived from an idealized model of the data (for genes without alternative splicing and no amplification bias), this relationship suggests that as the depth-coverage of a dataset increases, so does the likelihood of concordant expression, and hence the accuracy of HapTree-X. Figure 5-2 displays the theoretical curves depicting the exponential growth of likelihood of concordant expression as a function of coverage and $\beta$. We infer from this theoretical result that requiring a lower bound of DHE is beneficial for reliable DASE-based phasing given moderate coverage (30-50+). Furthermore, we present a table including minimum coverage required to obtain a probability of at least $1 - 10^{-\alpha}$ of concordant expression, given $\beta$.

![Figure 5-2: Likelihood of concordant expression (CE) as a function of coverage and differential haplotypic expression $\beta \in \{.55, .6, .65, .7, .75, .8, .85\}$]


<table>
<thead>
<tr>
<th>$\beta \setminus \alpha$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>.85</td>
<td>9</td>
<td>15</td>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>.8</td>
<td>13</td>
<td>21</td>
<td>31</td>
<td>41</td>
</tr>
<tr>
<td>.75</td>
<td>19</td>
<td>33</td>
<td>49</td>
<td>63</td>
</tr>
<tr>
<td>.7</td>
<td>31</td>
<td>55</td>
<td>79</td>
<td>105</td>
</tr>
<tr>
<td>.65</td>
<td>57</td>
<td>101</td>
<td>147</td>
<td>193</td>
</tr>
<tr>
<td>.6</td>
<td>133</td>
<td>235</td>
<td>339</td>
<td>447</td>
</tr>
<tr>
<td>.55</td>
<td>539</td>
<td>951</td>
<td>1377</td>
<td>1811</td>
</tr>
</tbody>
</table>

**Figure 5-3:** Coverage needed to obtain likelihood $1 - 10^{-\alpha}$ of concordant expression given a differential haplotypic expression of $\beta$ and an assumed opposite allele error rate of 2%.

### 5.3.2 Experimental results

#### Datasets and experimental setup

We evaluate haplotype reconstruction performance of HapTree-X on diploid RNA-seq and DNA-seq read datasets from GM12878, a well-studied lymphoblastoid cell line from a human female individual (NA12878) with European Ancestry (1000 Genomes Project [61]).

To assess the accuracy of phased haplotype blocks generated by HapTree-X, we compare our phasing results to a high-quality trio-phased SNP annotation of NA12878 (1000 Genomes Project Phase I), the gold-standard phasing reference. RNA-seq raw read datasets of GM12878 are obtained from ENCODE CSHL Long RNA-seq (wgEncodeCshlLongRnaSeq) [112] track with average sequencing depth of 100 million mate-pairs (2x76bp), transcriptome fragments sequenced from the nucleus with Poly-A+ and Poly-A− profiling.

For each RNA-seq dataset, we performed 2-pass alignments using STAR aligner v2.4.0d [113] by initially aligning raw reads to hg19 reference genome and then re-aligning reads to a second index generated from the splice junctions inferred from the first alignment.

We restricted DASE-based phasing within HapTree-X only to the SNPs that are located within the same gene in the GENCODE gene annotation v19 (wgEncode-
For joint DNA-seq and RNA-seq phasing experiments, we obtained genome sequencing reads of NA12878 from 1000 Genomes Project Pilot 2 release aligned to the hg19 reference genome using bwa aligner [40] and input both genome and transcriptome reads to the HapTree-X haplotype reconstruction framework.

We compared our results to those of HapCUT v0.7 sequence-based haplotype reconstruction tool [97]. To accommodate for long range splicing-junctions within RNA-seq read alignments, we defined maximum insert-size (maxIS) parameter to be longer than each chromosome's length.

Results from GM12878 datasets

In the RNA-seq read datasets from GM12878 (PolyA+ and PolyA- together), we observe that majority of the reads (≈ 89%) only cover a single heterozygous SNP in the genome. The distribution of read sizes are given in Table 5.1. Of the 19782889 reads containing one SNP, we are able to confidently assign expression biases to 675892 of them; we use these reads to phase, increasing the total number of reads to be used in phasing by 28%.

Table 5.1: Distribution of number of heterozygous-SNPs overlapped by each read in GM12878 RNA-seq data

<table>
<thead>
<tr>
<th>Read Size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8 - 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>19782889</td>
<td>2027207</td>
<td>290489</td>
<td>47424</td>
<td>17176</td>
<td>11941</td>
<td>10623</td>
<td>9119</td>
</tr>
<tr>
<td>%</td>
<td>89.12</td>
<td>9.133</td>
<td>1.311</td>
<td>.2137</td>
<td>.0774</td>
<td>.0538</td>
<td>.0479</td>
<td>.0411</td>
</tr>
</tbody>
</table>

Table 5.2 summarizes the haplotype reconstruction performance of HapTree-X in comparison to a contig-based algorithm, HapCut. Running HapTree-X without any DASE-based phasing (using only reads covering at least two SNPs) yields identical statistics (besides switch error) to HapCut, as both employ the ReadGraph structure to determine the SNPs and blocks to be phased. The switch error rate of HapTree-X without DASE-based phasing is consistent with that from with DASE-based phasing.
Results indicate that incorporating differential allele-specific expression in haplotype phasing increases the total number of SNPs phased, without increasing the switch error rate (with respect to the trio-phased gold-standard annotation). Furthermore, HapTree-X reduces the total number of blocks while increasing their overall sizes. We represent this by \( \#\text{Edges} = \#\text{SNPs} - \#\text{Blocks} \), equivalently the total number of pairs of adjacent (within a block) phased heterozygous-SNPs. This is also demonstrated by the large increase of total phased SNP pairs (any two SNPs within the same block). This indicates that HapTree-X produces longer haplotype blocks as a result of DASE-based phasing, as desired.

**Table 5.2:** Haplotype reconstruction results from HapTree-X and HapCut using DNA-seq and RNA-seq datasets from NA12878. Both HapCut and HapTree-X results are reported on RNA-seq read datasets as well as DNA-seq and RNA-seq merged datasets. DASE-based phasing only results from HapTree-X are also reported. For each dataset we report total number of phased SNPs, switch errors, haplotype blocks, edges and SNP pairs.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Stats</th>
<th>SNPs</th>
<th>Switch Errors</th>
<th>Blocks</th>
<th>Edges</th>
<th>SNP Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HapTree-X (DNA &amp; RNA)</td>
<td>979181</td>
<td>3767</td>
<td>298637</td>
<td>680544</td>
<td>5121692</td>
<td></td>
</tr>
<tr>
<td>HapCut (DNA &amp; RNA)</td>
<td>978811</td>
<td>5718</td>
<td>298710</td>
<td>680101</td>
<td>5101488</td>
<td></td>
</tr>
<tr>
<td>HapTree-X (RNA)</td>
<td>220849</td>
<td>641</td>
<td>88355</td>
<td>132494</td>
<td>412534</td>
<td></td>
</tr>
<tr>
<td>HapCut (RNA)</td>
<td>220386</td>
<td>669</td>
<td>88403</td>
<td>131985</td>
<td>380718</td>
<td></td>
</tr>
<tr>
<td>HapTree-X (DASE only)</td>
<td>1580</td>
<td>6</td>
<td>435</td>
<td>1145</td>
<td>4884</td>
<td></td>
</tr>
</tbody>
</table>

As discussed in 5.2.4 *Likelihood of concordant expression*, the solution of maximum likelihood (for any gene \( g \)) corresponds to that with concordant expression at all SNP loci within \( g \). HapTree-X therefore uses a threshold \( \lambda \) (negative log-likelihood of concordant expression) which requires any SNP to be concordantly expressed with probability at least \( 1 - e^{-\lambda} \), in order to be phased. We run HapTree-X while varying this threshold \( \lambda \); we compute the percentage of concordantly expressed SNPs and the total phased SNPs as we increase this threshold. As the threshold increases, HapTree-X demands any SNP to be phased to have a correspondingly high likelihood of concordant expression; as a result, the phasing accuracy of HapTree-X increases.
The cost paid for this increase in accuracy is a decrease in the total number of SNPs phased, as seen in Figure 5-4.

For the results reported in Table 5.2, we used a threshold value of 20. In theory, this threshold value $\lambda$ would produce a percentage of concordantly expressed SNPs equal to $1 - e^{-\lambda}$; however because of the structural noise commonly observed in aligned RNA-seq data due to false mapping, RNA-editing, as well complex alternative splicing events, we require a $\lambda' > \lambda$ to meet desired accuracy levels. Additionally, we require an estimated $\beta \geq .6$ for any gene to be phased using DASE, for motivation see Figure 5-2. Finally, we have several methods for managing alternative splicing events. HapTree-X can (1) avoid all genes with alternative splicing, (2) phase $s, s'$ only if the set of isoforms containing $s, s'$ are equal, and (3) phase independent of isoforms but require $s, s'$ to have coverage and DASE that are sufficiently similar. (3) was used in Table 5.2; (2), and especially (1), result in higher accuracy for lower $\lambda$, but of course phase fewer SNPs.

![Graph](image)

**Figure 5-4**: Rate of concordantly expressed SNPs (purple) and number of SNPs phased (green) by HapTree-X, as a function of $\lambda$ (negative log-likelihood of concordant expression).
Chapter 6

Conclusion and Discussion

In this thesis, we have presented a number of computational solutions for problems that arise in scalable storage, processing and analysis of sequencing datasets. However, the field of computational genomics is rapidly changing with new sequencing technologies, refinements in data generation, as well as new discoveries and developments in biology and computing.

Below we present ideas and discussions upon the computational approaches presented in this thesis with regards to the upcoming changes we expect in computational genomics.

6.1 Compression

The lossy quality score compression algorithm that we present in Chapter 3 is a novel approach that leverages ‘big data’ (i.e. population-level sequencing datasets from large studies) in order to perform faster and improved compression of sequencing datasets while at the same time providing better accuracy in downstream genotyping. Arguably, a fourth advantage of this method over existing approaches is its simplicity\(^1\): Quartz is based on a very simple algorithm that queries Hamming-1 neighbors of 32-mers sampled from the reads in the corpus. This corpus that stores the high-frequency 32-mers is built using a simple memory-mapped hash-table that supports very fast

\(^1\)The primary reason behind Quartz’s runtime efficiency.
k-mer queries. An augmented version of this hash-table can actually be used to build a fast alignment-free genotyping method as described in Shajii et al. [115].

Since our compression algorithm relies only on homologies between the k-mers in the read and the k-mers in the corpus, expanding this approach to emerging technologies, such as longer read lengths, linked-read sequencing technologies, or other species or sequencing dataset types (such as exome-sequencing, RNA-seq sequencing, CHIP-seq, etc.) is fairly straightforward as long as there is a reliable corpus generated from similar type of datasets from the same species. However, for technologies with higher sequencing error rates as well as datasets with high diversity, the balance between runtime, compression rate and downstream accuracy will need to be handled more carefully. Below, we provide a discussion upon these considerations.

Quality score compression for higher sequencing error rate technologies and high-diversity sequencing datasets

While Quartz provides superior compression rates with improved downstream accuracy for low sequencing error rate technologies like Illumina sequencing; sequencing technologies with higher error rates such as Pacific Biosciences’ single-molecule real-time sequencing [116] and Oxford Nanopore’s nanopore sequencing datasets [117], which have error rates around 10%, will not significantly benefit from Quartz quality score compression, due to low likelihood of matches between the k-mers in the reads and the k-mers in the corpus. Even for Ion Torrent [118] sequencing technology where the sequencing error rate is around 1.78% per base [119], around 10% of the k-mers in the read dataset will not have a match in the corpus within a Hamming distance of 1, noticeably hampering the homology based compression of Quartz.

One way to tackle quality score compression in high-sequencing error rate datasets is to perform larger distance searches from the k-mers in the reads to the reference. One issue that would arise from this change is the computational cost of finding longer-distance matches. Quartz operates through searching $3 \times k + 1$ k-mers within the Hamming-1 neighborhood of the read k-mer in the corpus. Increasing this neighborhood to Hamming-2 distance would add around 4500 more k-mers to query and
a Hamming-3 neighborhood will add around another 130000 k-mers. For this reason, the k-mer homology search approach within Quartz will need to be modified to accommodate these changes, potentially through a seed-and-extend based method which will be much slower than Quartz’ efficient k-mer hash-table. Another issue is that increasing our search neighborhood introduces higher likelihood of spurious k-mer matches. In order to prevent this, we should also increase the k-mer length that would further increase the computational cost of homology search, and also increase the coverage needed to generate a reliable high-frequency k-mer corpus.

A similar problem to the high sequencing error rate appears in the case of high-diversity species (or groups of species commonly sequenced together, like gut microbiome samples [120, 121]). Even for sequencing technologies with very low error-rates, the diversity within the species will make it difficult to find k-mer matches between the sequenced sample and the corpus. One way to fix this problem would be to build the corpus from a substantially larger sample (e.g. from 1000s of sequencing datasets instead of 100), which will increase the odds of finding k-mer matches between the newly sequenced sample and the corpus that contains k-mers with a certain level of coverage. In the case where there are very few k-mer matches within a corpus generated from the same species, it will primarily affect the compression rate and will not decrease the downstream accuracy of analyzing the newly sequenced individual. Once Quartz generates smoothed quality scores, additional lossy quality score compression methodologies (e.g. alphabet reduction, neighbor smoothing) can be used to achieve further compression, though it will likely reduce the downstream accuracy benefit gained from Quartz.

6.2 Compressive acceleration

The compressive read mapping framework that we present in Chapter 2 represents an advance for scalable processing of next-generation sequencing datasets. As read mapping is typically the costliest step in NGS analysis pipelines, substantial improvements to existing mapping approaches will also substantially accelerate sequence analysis
studies on large genomic datasets. CORA’s faster and more accurate alignments for multi-mapping allow for improvements in genotyping (repeat region analysis, structural variation and SNP detection, and copy-number variation analysis).

Furthermore, CORA’s relative runtime advantage over existing mappers increases substantially with both read depth-coverage and additional individuals in the dataset because of the compressive representation that enables much less time to be spent per read as the redundancy increases. In particular, if multiple individuals are included within a dataset, CORA’s compressive framework ensures their reads can be processed all at once, as opposed to separately for each individual. In this instance, CORA can reuse mapping computations performed for previous individuals in the dataset in order to avoid redundant calculations in the additional individuals. CORA’s relative runtime improvements will also substantially improve as sequencers generate higher-quality reads, as its compressive framework achieves gains inversely related to the sequencing error rate, which decreases as sequencing technologies improve.

The flexibility of the CORA framework enables it to adopt the functionalities of the coarse mappers that it uses. For example, CORA can perform alignment with indels, provided that the coarse mapper used within CORA is able to report coarse mappings with indels. Other coarse mapper functionalities can also be incorporated into the CORA framework without a large-scale implementation update, such as SNP-aware alignment algorithm of mrsFAST-Ultra, which reduces reference mapping bias, or dynamic trimming of read ends that allow low quality portions of reads to be trimmed during alignment [122]. Similarly, CORA’s indel detection capabilities can be improved by utilizing other mappers’ capabilities to capture longer indels during the coarse mapping stage.

**Extensions to other sequencing data types and alignment distance metrics**

Though CORA is designed to accelerate mapping paired-end Illumina sequencing reads with relatively low sequencing errors, it is possible to extend the framework for sequencing technologies with higher error rates (e.g. Pacific Biosciences and Oxford Nanopore), using three different strategies: introducing higher error rate homology
blocks in the homology table, approximate compression of k-mers in the read dataset, or sampling shorter and more k-mers per read. However, the ideal combination of these for an optimal trade-off between memory, runtime and mapping sensitivity is not yet established.

In the case of linked-read sequencing [123] (where reads that share a barcode are highly likely to come from the same molecule), or chimera and structural variation detection (where mate-pairs may be far apart, inverted, or on different chromosomes altogether), CORA framework can be run as normal until the stage where mate-pairs are merged. Since CORA generates all k-mer neighbors during the homology table traversal stage, the information needed for solving these alternative mapping tasks are already present in the final mate-pair merging stage. Depending on the analysis type, different merging procedures can be applied. For example in the case of chimera detection, the merging criteria could be relaxed by allowing mates that are mapped to different chromosomes. For structural variation detection, we can relax the constraints of the mapping direction of paired-end reads as well as the distance between them. In the case of linked-read sequencing, all reads that share a barcode can be traversed simultaneously, and the merging stage could be updated with the probabilities associated with the barcode-sharing reads being aligned to the same region of the genome.

Compressive acceleration of transcriptome sequencing datasets (RNA-seq) is more challenging, yet also more rewarding due to higher likelihood of ambiguous spliced mappings [124]. In the case that the gene model as well as all valid splice junctions are known beforehand, an artificial chromosome can be constructed containing \((2 \times k - 1)\)-mers from each junction and k-mers that span splice junctions can be captured through coarse-mapping to the artificial chromosome or through homology table traversal. The k-mer merging stage will also need to be updated for handling merges between regular chromosomes and the artificial chromosome. In the case that the splice junctions are not known before hand; however, CORA will need to employ a more involved mapping recovery stage where k-mers that span splice junctions are handled individually using neighboring k-mers in the read as anchors. Some caveats
would be the added runtime of costlier mapping recovery stage as well as short exons that overlap with multiple $k$-mers in the read. The latter issue can be alleviated to some extent by sampling overlapping $k$-mers from the read, though this approach will likely further reduce the speed gains from compressive acceleration.

6.3 Haplotype reconstruction

In Chapter 4, we present an efficient algorithm, HapTree, for polyploid haplotype reconstruction using NGS sequencing data and a new metric, vector error rate, for measuring accuracy for phased polyploid haplotypes against a ground-truth dataset. HapTree identifies phases that maximize our RL metric, a relative likelihood function which measures the quality of a given phase according to the read dataset. We demonstrate the advantages of such a likelihood formulation over the existing MEC (minimum error correction) score metric in phasing polyploid and diploid genomes. Our results indicate that HapTree can be used in phasing individual triploid and tetraploid genomes, as well as diploid genomes with better accuracy compared to existing methods. HapTree can also scale to genomes of higher ploidy. We anticipate HapTree to be useful for the investigation of the role of heterozygosity in plant, fish, and other species.

In Chapter 5, we present a novel haplotype reconstruction framework, HapTree-X, that is tailored towards transcriptome sequencing datasets, leveraging both sequence contiguity information within reads that overlap multiple heterozygous-SNPs as well as differential allele-specific expression (DASE). We present a maximum likelihood model for DASE-based haplotype reconstruction and a computational phasing algorithm that integrates haplotype reconstruction through differential expression and sequence contiguity in a joint Bayesian model.

Future directions for HapTree-X framework include incorporation of complex alternative splicing events for more accurate estimation of DASE within an augmented mathematical model of isoform-specific differential allelic expression within genes. We anticipate HapTree-X to be useful for personalized medicine through identification of
disease association of gene haplotypes.
Bibliography


184


