# Methods for Identifying Variation in Large-Scale Genomic Data

by

M. Jacob Pritt

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

September, 2018

# Abstract

The rise of next-generation sequencing has produced an abundance of data with almost limitless analysis applications. As sequencing technology decreases in cost and increases in throughput, the amount of available data is quickly outpacing improvements in processor speed. Analysis methods must also increase in scale to remain computationally tractable. At the same time, larger datasets and the availability of population-wide data offer a broader context with which to improve accuracy.

This thesis presents three tools that improve the scalability of sequencing data storage and analysis. First, a lossy compression method for RNA-seq alignments offers extreme size reduction without compromising downstream accuracy of isoform assembly and quantitation. Second, I describe a graph genome analysis tool that filters population variants for optimal aligner performance. Finally, I offer several methods for improving CNV segmentation accuracy, including borrowing strength across samples to overcome the limitations of low coverage. These methods compose a practical toolkit for improving the computational power of genomic analysis.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Motivation

## 1.1 The Rise of Sequencing Technologies and Data

Genomic sequencing has made great strides since its painstaking and expensive beginning in the 1970s. The completion of the Human Genome Project [1, 2] marked a major milestone, but it was not until several years later that the first high-throughput sequencing platform ushered in the era of next-generation sequencing. The subsequent explosion in sequencing techniques includes both short and long-read, paired-end and unpaired technologies, and its applications extend to sequencing messenger RNA (RNA-seq), mapping DNA methylation, and measuring transcription factor binding (ChIP-seq) [3]. The recent advent of single-cell sequencing allows the study of varia-

tion within systems at a cellular level.

In tandem with the rise of sequencing technologies, the amount of sequencing data has also exploded at an exponential rate and shows no sign of slowing [4]. Large-scale efforts such as HapMap [5], the 1000 Genomes Project [6] and UK10K [7] have genotyped thousands of human genomes. Non-human data also is increasing at a rapid pace; similar large-scale efforts exists for many plant strains, such as the 1001 Genomes Project for *Arabidopsis thaliana*, and for metagenomics, such as the Earth Microbiome Project. In this thesis I focus primarily on human sequencing data, though the methods extend easily to other genomes.

The traditional pipeline for processing sequencing data (both DNA and RNA) involves first aligning reads. If a reference genome exists, alignment tools like Bowtie [8] or BWA [9] provide the foundation for all downstream analysis. These tools use a precomputed index to place seeds on the reference genome and then find the exact start position and gapped alignment with dynamic programming. Aligned reads are generally stored in Sequence Alignment/Map (SAM/BAM) files [10], a format which stores all relevant read information including location, CIGAR string, mate information, sequence, and quality string. This alignment file is the input for a host of downstream analysis tools.

The "alignment against a linear reference" pipeline, both versatile and simple, has long been the de facto method for processing sequencing data. However, the recent increase in sequencing data has triggered the design of alternative methods for each

stage in this pipeline. Among the most significant, pseudoalignment methods replace the rigor of exact alignment with approximate mapping. The graph genome is an extension of the linear reference genome to the pan-genome to eliminate sources of bias and inaccuracy in mapping.

### Pseudoalignment Methods

Pseudoalignment methods originate from the observation that exact alignments are not necessary for many downstream analyses. These tools sacrifice exact alignment position and gap information to skip the costly exact mapping step and large alignment output files. Sailfish [11] and kallisto [12] both utilize pseudoalignment to provide quantitation estimates in a fraction of the traditional alignment time. Similarly, Salmon [13] does not compute gapped alignments but is not strictly pseudoalignment since it tracks read position and orientation. The benefits of pseudoalignment are two-fold; first, alignment is significantly faster, and second, large alignment files are not produced. I focus here on the elimination of traditional alignment files.

A drawback of SAM/BAM, and of any format that stores data on a per-read basis, is that file size grows close to linearly with the number of reads. As sequencing continues to improve, and as public archives fill with more datasets, the burden of storing aligned sequencing data also increases rapidly. The Sequence Read Archive [14], which stores raw sequencing reads, grew from 4 to almost 20 petabases from August of 2015 to August of 2018. It is increasingly common for RNA-seq studies to

span hundreds or thousands of samples, with tens of millions of reads per sample [15, 16].

In Chapter 2 I present a novel compression method for SAM/BAM data based on the principles of pseudoalignment. This compression method, implemented in Boiler [17], discards precise read alignment information and instead converts them to easily-compressible coverage vectors with a few accompanying empirical distributions. Boiler compresses alignment datasets to roughly the same size as their BigWig counterparts but like pseudoalignment methods preserves relevant information for accurate downstream isoform assembly and quantitation.

### Graph Genome

A second limitation of traditional alignment tools like Bowtie and BWA is that they rely on a linear reference genome representing a single individual (although it may be composed of many disjoint contigs from different individuals.) As high-throughput technologies have brought an explosion of population genetics information, the question is emerging: how can we use population genetics information to improve accuracy of genomic analyses? This has fueled interest in techniques that depart from a linear string as point of reference for all individuals, and toward pan-genome representations [18, 19] more inclusive of genetic variation.

While methods for including variants in the reference are growing in number [20, 21, 22, 23, 24, 25, 26], there is little or no work on how to choose which variants

to include. The set of variants must be filtered for two reasons; first, current graph genome alignment methods are often unable to index a graph containing all known variants; for example, HISAT2 exceeds 1TB of memory when building a human genomic index containing even 10% of 1000 Genomes variants. Secondly, many variants introduce additional ambiguity when added to the graph, asserting a negative force on alignment accuracy. These considerations, which will continue to be exacerbated by increasing amounts of data, necessitate some level of filtering on graph variants. Past studies have made such decisions in ad hoc ways, with some filtering according to allele frequency [27, 22], ethnicity [21], or both [23].

In Chapter 3, I examine the advantages and disadvantages of adding variants to the reference. I show that the disadvantages are important to measure, since simply adding more variation to the reference eventually reduces alignment accuracy. I suggest efficient models for scoring variants according to the effect on accuracy and "blowup" (computational overhead), and further show that these scores can be used to achieve a balance of accuracy and overhead superior to current approaches. For example, extrapolating to a whole-human DNA sequencing experiment at 40-fold average coverage, I estimate that a well-engineered augmented reference can yield about 4.8M more correctly aligned reads and 1.2M fewer incorrectly aligned compared to the linear reference. The methods for selecting variants also reduce reference bias, a chief goal of graph genomes. Finally, I compare the accuracy yielded by my methods to that achieved using an ideal personalized graph genome. I show

that these methods approach the ideal much more closely than both linear genomes – even when they are modified to contain only major alleles – and graph genomes built on different sets of variants.

These methods are implemented in a new open source software tool called FORGe [28]. I demonstrate FORGe in conjunction with the HISAT2 [26] graph aligner and with another aligner based on the Enhanced Reference Genome [21]. But FORGe's models and methods are suitable for any aligner that can include variants in the reference.

**Processing large-scale single-cell data**

Single-cell sequencing [29] is emerging as a critical technology for understanding the biology of cancer [30], neurons [31], and other complex systems [32]. Studying genomic variation at the single-cell level allows investigators to unravel the genetic heterogeneity within a sample and enables the phylogenetic reconstruction of subpopulations beyond what is possible with standard bulk sequencing, which averages signals over millions of cells. To date, many thousands of individual human cells have been profiled to map the subclonal populations within cancerous tumors [33, 34], and circulating tumor cells [35], discover mosaic copy number variations in neurons [31], and identify recombination events within gametes [36], among many other applications.

One of the most significant applications of single-cell sequencing is to identify

large-scale (>10kb) copy-number variations (CNVs) [31, 33]. These events can radically alter cell biology and are important in neurodegenerative and developmental disorders among many other conditions. Within heterogeneous samples, CNVs form a genetic fingerprint from which the phylogenetic history of a sample may be inferred [37]. For example, within cancer biology, this technique has been used to study the widespread heterogeneity within tumors and has led to greater understanding of tumor development and metastasis [33, 34]. CNVs are also the most readily accessible variants to analyze by single-cell sequencing [38]. Whole-genome amplification (WGA) techniques are needed to prepare the DNA within a single cell for high-throughput sequencing. Despite dramatic improvements since their introduction, WGA protocols inevitably have variable amplification efficiency across the genome and even total loss of coverage in certain regions [38]. This makes detecting single-nucleotide mutations and other small mutations unreliable, although large copy-number variations (>10kbp) are robustly detected through statistical analysis of changes in read depth across the genome.

Given the insight derived from single-cell sequencing, many researchers are now interested in applying the technology to diverse cell types and species. However, the downstream analysis is complex, requiring a multistage pipeline to identify copy number variants from low coverage single cell sequencing data. Addressing this critical need, *R. Aboukhalil et al.* previously introduced Ginkgo [39], an open-source web platform for the interactive analysis and quality assessment of single-cell copy-

number alterations. Ginkgo fully automates the process of binning, normalizing, and segmenting mapped reads to infer copy number profiles of individual cells, as well as constructing phylogenetic trees of how those cells are related. Ginkgo was validated by reproducing the results of five major single-cell studies, and also by examining the data characteristics of three commonly used single-cell amplification techniques: MDA, MALBAC, and DOP-PCR/WGA4 through comparative analysis of 9 different single-cell datasets.

In chapter 4 I present a set of user interface and accuracy improvements to Ginkgo in light of the increasing scale of single-cell sequencing experiments and soon to be released in Ginkgo 2. These improvements include a novel binning technique to avoid unmappable regions more robustly, a bin refinement step to localize copy number changes more accurately, and a re-segmentation step that borrows strength from replicate cells such as multiple cells sampled from the same clone within a tumor.

Overall, this work aims to provide tools for a future where personalized genomic sequencing is commonplace; where sequencing is not only targeted to specific systems and cells, but results can be analyzed in the context of comprehensive population data. The methods presented here help to establish the next step in effective and accurate analysis.

# Chapter 2

# Radically Lossy compression of RNA-seq alignments with Boiler

In this chapter, I present a novel lossy compression method for alignment data examine its effect on downstream isoform assembly and quantitation. I implemented this method in Boiler, originally published in 2016 [17].

## 2.1 Introduction: Compression Methods for Alignment Data

Compressed formats for SAM/BAM files eliminate redundant data across reads or alignments, decreasing file size and allowing size to grow sub-linearly (rather than linearly) with the number of reads. CRAM [40], NGC [41], Goby [42] and REFEREE

[43] use reference-based compression, which was proposed earlier [44, 45], to replace a read sequence with a concise description of how it differs from a substring of the reference. Quip [46] uses arithmetic coding together with a sequence model trained on-the-fly to compress losslessly and without a reference. Goby uses a range of strategies, including column-wise compression and detailed modeling of relationships between columns. REFEREE uses separable streams and clustering of quality strings. In these formats, the alignments and the fields are largely preserved, but are compressed along with neighbors row-wise (together with the other fields of the same alignment), or column-wise (with other instances of the field across alignments).

These studies also explore lossy compression schemes, in which data not used in certain analyses, such as read names and quality strings, is selectively discarded. Many tools optionally discard read names and quality values, and REFEREE clusters quality strings and replaces each with a single representative from its cluster.

Boiler takes a radically lossy approach to compressing RNA-seq alignments, yielding very small compressed outputs. Inspired by the notion of transform coding, Boiler converts alignment data from the "alignment domain," where location, shape and pairing information are stored for every alignment, to the "coverage domain," where the coverage vector is stored and alignment information is inferred where needed. Boiler keeps only a set of coverage vectors and a few empirical distributions that partially preserve fields such as `POS` (offset into chromosome) `TLEN` (genomic outer distance), `XS:i` (strand) and `NH:i` (number of hits). Consequently, Boiler is lossy in

an unusual sense: compressing and decompressing might cause alignments to shift along the genome, change shape, or become matched with the wrong mate. Table 2.1 presents a comparison of how CRAM, Goby, and Boiler preserve read information.

Boiler should not be considered a general-purpose alignment compression tool. Because it discards quality values and non-reference alleles, its output is not appropriate for downstream tools requiring such data, such as variant callers and tools for allele-specific expression. However, Boiler preserves the data most relevant to popular downstream RNA-seq tools for quantification, assembly and differential expression. We show that popular isoform-level tools – Cufflinks [47] and StringTie [48] – yield near-identical results for isoform assembly and quantification when the input is Boiler-compressed.

Boiler yields extremely small file sizes, more than 3-fold smaller than files produced by CRAM and Goby for paired-end samples with at least 10M paired-end reads. Unlike other compression tools, Boiler's compression ratio improves substantially as input file size grows, growing from about 10-fold for lower-coverage unpaired samples to over 50-fold for higher-coverage samples. Speed and memory footprint are comparable to other compression tools despite the fact that, as we show, recovering alignments from a coverage vector is computationally hard. Also, because nucleotide data is removed, Boiler-compressed data is effectively de-identified, making it easier to pass between parties securely.

Boiler also provides a range of speedy queries. Many compression tools provide a

way for the user to extract alignments spanning a particular genomic interval from the compressed file. REFEREE goes a step further by enabling faster queries when the user is concerned with only a subset of the fields. Boiler goes further still by providing fast queries that are directly relevant to downstream uses of RNA-seq data. Boiler allows the user and downstream tools to (a) iterate over "bundles" of alignments according to inferred gene boundaries, (b) extract the coverage vector across a genomic interval, and (c) extract alignments overlapping a genomic interval.

## 2.2 Methods

### 2.2.1 Compression

Boiler implements a lossy compression scheme that preserves only the data needed by downstream isoform assembly tools such as Cufflinks and StringTie. For this reason, read names and quality strings are discarded, along with other data that has little or no bearing on downstream RNA-seq analysis.

Given a set of alignments to a reference genome, Boiler first partitions the alignments into "bundles" of overlapping reads. Bundles are computed in the same manner as Cufflinks' initial bundling step: As sorted reads are processed, if the current read starts within 50 bases of the end of the current bundle, the read is added to the bundle. Otherwise, the current bundle is compressed and a new bundle is initialized beginning with the current read.

Boiler converts each bundle into a set of coverage vectors and tallies of observed read lengths. For most Illumina sequencing datasets, reads are uniform-length (or nearly so, e.g. after trimming), yielding a concise tally. If any alignments in the bundle are paired-end, Boiler also stores a tally of observed genomic outer distances as reported by the aligner in the `TLEN` SAM field. Note that `TLEN` includes the lengths of all the introns spanned by the alignment, so we refer to this as "genomic outer distance," rather than "fragment length." The coverage vector is compressed using run-length encoding, storing each run of the same coverage value as the value and length of the run. Run-length is particularly efficient in low-coverage regions.

Each bundle is compressed as follows:

1. Boiler scans the bundle's alignments to find splice sites spanned by at least one alignment. Boiler divides the portion of the genome spanned by the bundle into "partitions" formed by cutting at every splice site (Figure 2.1a).

2. Boiler assigns each alignment to a *bucket* according to: (a) the subset of partitions spanned by the alignment, (b) the value in the alignment's `NH:i` field, indicating the number of distinct locations where the read aligned to the reference, and (c) the value in the `XS:A` field, indicating whether spanned splice motifs are consistent with the sense (`+`) or anti-sense (`-`) strand of the gene. Alignments not spanning a junction usually lack the `XS:A` field; Boiler treats these as though the `XS:A` field contains a "dummy" value indicating the strand is unknown.

3. For each bucket, Boiler computes the coverage vector from the alignments assigned to it. Boiler writes the run-length encoded coverage vector (Figure 2.1b) followed by the read and genomic outer distance distributions (Figure 2.1c) for the junction.

Each bundle, which consists of many buckets, is compressed independently using the DEFLATE algorithm as implemented in the `zlib` package from the Python Standard Library. Each bundle is compressed separately to make targeted queries efficient, as discussed in the "Queries" section.

Some RNA-seq alignment tools (including HISAT [49]) output SAM records for reads or ends that fail to align, whereas others (including TopHat 2 [50]) do not. Boiler deals only with aligned reads. SAM records for unaligned reads are ignored, and those reads are not represented in a compressed Boiler file. Additionally, if one end of a paired-end read is "orphaned" – i.e. its opposite end fails to align – Boiler will convert the orphan to an unpaired read. The paired nature of orphaned reads is lost during Boiler compression.

### 2.2.2 Unbundled alignments

Prior to compression, Boiler must identify and handle paired-end alignments that span bundles in unexpected ways. We call these *unbundled* alignments. Unbundled alignments fall into four categories: (a) one end falls within an intron spanned by the other end, (b) the two ends align to different chromosomes, (c) the two ends align to

the same chromosome but very far from each other, (d) one end is assigned to the sense strand, while its mate is assigned to the anti-sense strand. Both TopHat and HISAT report such alignments, though they constitute only a small fraction of the alignments in a typical dataset.

These alignments are hard to fit into the bundling scheme described previously. Reads in category (a) are biologically implausible. Boiler treats them as unpaired reads by default, however the user may choose to preserve these pairings.

Categories (b) and (c) could be scientifically relevant and should be preserved. For instance, alignments in category (b) may be evidence of gene fusions. Boiler stores all alignments in categories (b) and (c) in a special "unbundled alignments" section of the compressed file. Unbundled alignments are stored in *bundle-spanning buckets*. A bundle-spanning bucket is identical to a normal bucket, but includes the indices of the two bundles it spans in addition to the list of partitions spanned from each bundle. The bundle-spanning buckets are stored as a contiguous list, compressed in small chunks using zlib, and indexed to reduce work for targeted queries.

Treatment of pairs in category (d) is configurable by the user. By default, they remain paired and one end, selected at random, is modified to match the other's strand. Optionally (using `--split-discordant`), such pairs can be treated as unpaired reads, which is consistent with how they are treated by Cufflinks and StringTie.

### 2.2.3 Multi-mapping reads

RNA-seq reads may align equally well to many genomic locations. Such reads are called "multi mappers." Because downstream tools might treat multi mappers differently from uniquely mapped reads, at least some multi-mapping information must be preserved by Boiler.

For a multi-mapping read, Boiler preserves each alignment along with its NH:i extra field. However, Boiler also discards read names. This can have an adverse impact on downstream tools that rely on read names to establish the one-to-many relationship between reads and their alignments. For example, Boiler does not distinguish between primary and secondary alignments, so algorithms that assign special weight to primary alignments will suffer from compression. Our fidelity experiments show that discarding read names does not substantially impact the accuracy of downstream tools for isoform assembly and quantification. However, it does have an adverse effect on quantification accuracy when Cufflinks is asked to quantify from a given gene annotation (`-G` mode). StringTie's accuracy when quantifying from a given gene annotation (`-G -e` mode) is not substantially affected.

Boiler might reduce the adverse effects of discarding read names by splitting primary and secondary alignments into separate buckets, at the cost of worse compression ratios. Though promising, we do not explore this option in this manuscript.

| SAM Feature | CRAM | | Goby | | Boiler |
|---|---|---|---|---|---|
| | Default | Config. | Default | Config. | |
| Read Name | Yes[1] | No | No | Yes | No |
| Flags | Yes | No | Yes | No | No |
| Mapping Quality | Yes | No | Yes | No | No |
| Read Sequence | Yes | No | Yes | No | No |
| Quality Scores | No | Yes | Yes[2] | Yes | No |
| Tags | No | Yes | MD | Yes[3] | XS, NH |

Table 2.1: Comparison of the SAM fields stored by different compression tools. CRAM and Goby can preserve some fields through configurable options, summarized in the "Config" columns.
[1]CRAMtools documentation claims that by default read names should not be preserved, however we were not able to replicate this functionality.
[2]For mismatches only
[3]Goby preserves either all tags or just the MD tag.

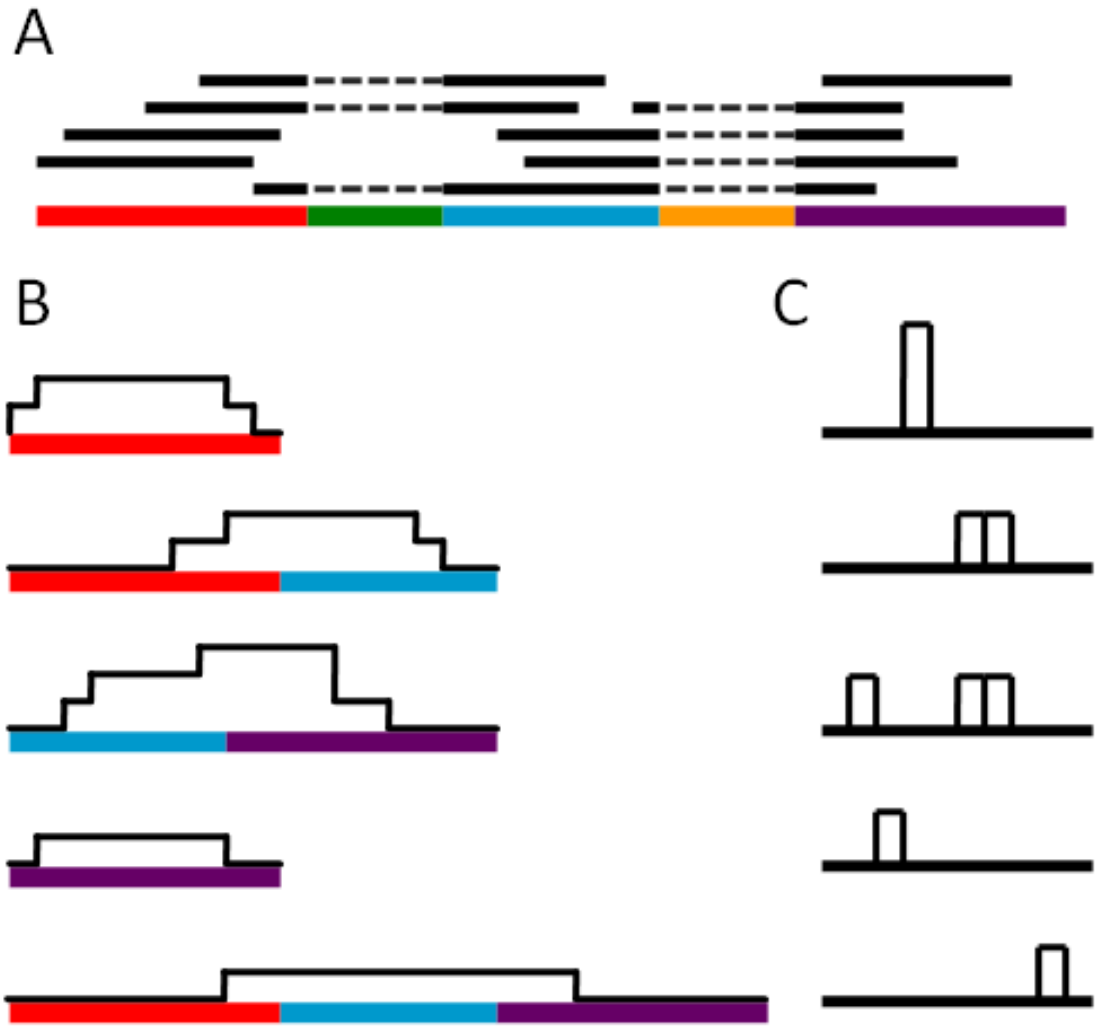


Figure 2.1: Illustration of how Boiler compresses alignments in a bundle, for a dataset with unpaired reads. (a) The genome is divided into "partitions" (colored segments) based on the processed splice sites. A bucket is defined by the subset of partitions spanned (as well as the values of the NH:i and XS:A fields, though these are omitted from the figure for simplicity). Each bucket stores (b) the coverage vector for the partitions spanned and (c) a histogram of the lengths of all reads assigned to the bucket.

### 2.2.4 Decompression

To decompress, Boiler first expands each bundle with the INFLATE algorithm as implemented in the Python `zlib` module, then expands each bucket.

When decompressing a bucket, Boiler seeks to recreate the set of alignment intervals that yielded the bucket's coverage vector and read and genomic outer distance tallies. This is a two-step process; first reads must be recovered from the coverage vector and read length tally ("read recovery"), then the recovered reads must be paired according to the paired length tally ("pairing").

The *read recovery* problem may not have a unique solution; e.g., consider a compressed dataset with read lengths $l_1$ and $l_2$ ($l_1 \neq l_2$) and a coverage vector containing 1 at all positions in the range $\big[0, l_1 + l_2\big)$. This case has two valid solutions:

$r_1 = \big[0, l_1\big)$, $r_2 = \big[l_1, l_1 + l_2\big)$

and

$r_1 = \big[0, l_2\big)$, $r_2 = \big[l_2, l_1 + l_2\big)$

Thus, we cannot guarantee perfect recovery of the compressed reads.

We define the read recovery problem as follows. Given a coverage vector and tally of read lengths, we seek a list of decompressed reads (genomic intervals) such that

1. the decompressed read lengths are a subset of those given in the tally,

2. at no position does the coverage vector produced by the decompressed reads

exceed the value in the original coverage vector, and

3. the sum of the lengths of all decompressed reads is maximized.

This formulation is general enough to tolerate an input where the read length tally and coverage vector are not compatible, i.e., where no solution fits both precisely. In this case, the algorithm might decompress only some of the reads in the input tally.

**Lemma 2.2.1** *The read decompression problem is strongly NP-hard.*

**Proof** Consider the Multiple Subset Sum Problem (MSSP), defined as follows. Given $n$ items with weights $w_1, w_2, ..., w_n$ and $m$ knapsacks with capacities $c_1, c_2, ..., c_m$, assign items such that:

1. each item is assigned to up to 1 knapsack

2. the capacity of each knapsack is not exceeded by the combined weights of the items assigned to it

3. the total weight of the items in all the knapsacks is maximized.

MSSP is known to be strongly NP-hard [51].

We reduce MSSP to a special case of the read decompression problem where the coverage vector never exceeds 1. We first construct a vector $C$ encoding knapsack capacities in unary. We start with empty $C$ then, for each $i$, append $c_i$ 1s followed by a single 0. Because the length of $C$ depends on the numeric knapsack weights, this is a pseudo-polynomial time reduction. Next, we let the read length tally equal the item

weight tally. Finally, we run our decompression algorithm on the coverage vector $C$ and read length tally. The algorithm packs reads into the nonzero stretches of $C$. This solution is converted to an MSSP solution by converting reads to the corresponding items and stretches of the coverage vector to the corresponding knapsacks.

The reduction satisfies the requirements of a pseudo-polynomial transformation [52]. Hence, the read decompression problem for unpaired reads is strongly NP-hard. □

We observe that the read recovery problem is NP-hard in general, but that some special cases are easily solved. When all reads are the same length, for example, the solution is unique and can be found efficiently. We also observe that second-generation sequencing produces datasets with uniform or near-uniform (e.g. after trimming) read-length tallies. These facts lead us to propose the greedy algorithm described below. The algorithm is not optimal in general, but it is well suited to cases where the input read lengths are uniform or almost uniform.

### Recovering Unpaired Reads and Read Ends

The algorithm works from one end of the coverage vector to the other, extracting reads that are "consistent" with coverage. A read is consistent with coverage if removing the read and decrementing the corresponding coverage-vector elements does not cause any vector elements to fall below zero. When a consistent read is selected for extraction, the corresponding coverage-vector elements are decremented and the

process repeats. The process stops when the far end of the coverage vector is reached.

When reads have uniform length, the algorithm yields the correct solution. When reads have various lengths, the problem is harder and the algorithm may fail to yield the optimal solution. In the case where reads are various lengths, Boiler's algorithm uses heuristics to arrive at a solution where (a) the coverage vector induced by the extracted reads matches the true vector as closely as possible, and (b) the distribution of extracted read lengths matches the true distribution of read lengths as closely as possible. Boiler favors (a) over (b); i.e. it will artificially lengthen or shorten the extracted reads to fix small coverage discrepancies.

The algorithm works from one end of the coverage vector to the other, removing reads that remain consistent with the coverage vector. We take advantage of the homogeneous read length distribution produced by sequencing experiments by preferentially removing reads of the most common length. When necessary, we adjust the lengths of previously found reads by a few bases to match the coverage vector as closely as possible.

Initially, we extract reads in end-to-end sets of the form $(a, b, n)$ where $a$ and $b$ are the starting and ending indices in the coverage vector and $n$ is the number of end-to-end reads. Each read set must satisfy

$$n \cdot l_{min} \leq (b - a) \leq n \cdot l_{max}$$

where $l_{min}$ and $l_{max}$ are the minimum and maximum lengths in the read distribution, respectively. Each time we find a new read $(b, c)$, we search for an existing read set

matching $(a, b, n)$ and update it to $(a, c, n + 1)$. If no such read exists, we add a new read set $(b, c, 1)$.

We define two helper functions $extend(x_0, x_1)$ and $shorten(x_0, x_1)$. $extend(x_0, x_1)$ searches for a read set of the form $(a, x_0, n)$ satisfying

$$n \cdot l_{min} \leq (x_1 - a) \leq n \cdot l_{max}$$

and updates it to $(a, x_1, n)$ and decrements the coverage vector in the range $[x_0, x_1)$ by 1. $shorten(x_0, x_1)$ searches for a read set of the form $(a, x_1, n)$ satisfying

$$n \cdot l_{min} \leq (x_0 - a) \leq n \cdot l_{max}$$

and updates it to $(a, x_0, n)$ and increments the coverage vector in the range $[x_0, x_1)$ by 1.

These functions allow us to adjust previous reads by small amounts to fit in later reads. The read extraction algorithm works as follows:

Let *start* and *end* be the indices of the first and last nonzero elements in the coverage vector, respectively. We find $a$ and $b$ such that $cov[i] > 0 \ \forall \ i \in [start, a)$, $cov[a] = 0$, and $cov[i] = 0 \ \forall \ i \in [a, b)$, $cov[b] > 0$.

Special end case: if $a = end < start + l_{min}$, we first attempt to run $extend(start, a)$. If unsuccessful, we decrement the bases in the coverage vector in the range $[start, a)$ but do not add a new read.

If $a \geq l_{mode}$, we add a new read $(start, start + l_{mode})$ and update the coverage vector.

Otherwise, we attempt to run $extend(start, a)$. If unsuccessful, we attempt to run $shorten(a, b)$. If this is also unsuccessful, we do one of the following:

1. If $(a - start) \geq l_{min}$, we add a new read $(start, a)$ and update the coverage vector.

2. If $\frac{l_{min}}{2} \leq (a - start) < l_{min}$, we add a new read $(start, start + l_{mode})$ and update the coverage vector.

3. If $(a - start) < \frac{l_{min}}{2}$, we decrement the bases in the coverage vector in the range $[start, a)$ but do not add a new read.

We then update *start* and *end* and repeat until the coverage vector is empty.

**Paired-end Read Recovery**

If the data contains paired-end reads, we must also solve the *read pairing* problem. We would like Boiler to restore the paired-end relationships in a way that matches the original genomic outer distances as closely as possible. We start with (a) a collection of reads (ends), all of which are initially unpaired, and (b) the "true" genomic outer distance tally, storing the frequencies of each distance, which was compiled and stored during compression.

We use the following greedy algorithm to pair up reads in a way that closely matches the true genomic outer distance tally. Each read is examined, working inward from the extremes, alternating between the left and right extremes. For each read,

we seek the most distant read such that the resulting pairing is compatible with distances remaining in the tally. When two reads are paired in this way, they are removed from future consideration, and the corresponding element of the tally is decremented. Reads that are not paired in this way are matched up randomly in a second pass.

### 2.2.5 Queries

Boiler allows the user to query a compressed RNA-seq dataset to (a) iterate over genomic intervals delimiting regions of non-zero coverage, roughly corresponding to genes, (b) extract the genomic coverage vector across a specified genomic interval, and (c) extract alignments overlapping a specified genomic interval. Because each bundle of alignments is compressed separately, Boiler can answer such queries without decompressing the entire file.

Bundle boundaries are stored in the index at the beginning of the compressed file, so skipping to a particular bundle can be accomplished with a single uncompressed index lookup. To compute the coverage query (query $b$, above), Boiler combines the relevant portions of the coverage vectors for all the buckets overlapping the specified region. This requires that Boiler decompress the DEFLATED and run-length encoded coverage vectors, but does not require the more work-intensive read and pair recovery algorithms. Alignment-level queries (queries $a$ and $c$ above) are more expensive, requiring Boiler to run the greedy read recovery algorithm on each of the buckets

overlapping the specified region.

Downstream tools like Cufflinks can be modified to query Boiler-compressed files directly, removing the need for an intermediate SAM/BAM file. When a downstream tool requires access only to information about gene boundaries (query *a*, above) or about targeted regions of the coverage vector (query *b*), Boiler's queries can be much faster than directly querying a sorted and indexed BAM file.

### 2.2.6 Implementation

Boiler is implemented in Python and is compatible with Python interpreters version 3 and above. All of the Python modules used by Boiler are in the Python Standard Library, making Boiler quite portable across Python installations and interpreters. For example, we use the fast PyPy interpreter for our experiments.

## 2.3 Results

We used Flux Simulator v1.2.1 [53] to simulate 10 RNA-seq samples from the BDGP5 build of the *D. melanogaster* genome and the Ensembl release 70 [54] gene annotation. We simulated both paired-end and unpaired RNA-seq samples for a series of library sizes: 0.5, 1, 2.5, 5, 10, and 20 million reads. We also simulated two samples from the hg19 build of the human genome and Gencode v12 gene annotation [55] containing 20 and 40 million paired-end reads. We also used two real human RNA-seq

samples. We used sample HG00100 from the GEUVADIS [15] study, consisting of about 20 million paired-end reads. We also chose one of the seven technical replicates of the 3:1 ratio of Universal Human Reference RNA to Human Brain Reference RNA from the SEQC study [56]. The study accession is SRP025982 and the individual replicates have run accessions SRR1216073 – SRR1216079. Each replicate consists of approximately 11-million paired-end reads. We tested sample SRR1216073, labeled "SRP025982" in the results below. All samples were aligned to the reference genome using either TopHat 2 v2.1.0 [50] with default parameters, or HISAT 0.1.6-beta [49] with default parameters. *D. melanogaster* samples were aligned to the BDGP5 reference and human samples were aligned to the hg19 reference.

We compared Boiler's speed, compression ratio, and peak memory usage to Goby and CRAMTools. Boiler and Goby remove read names by default, but CRAM does not. (CRAMtools has an option to preserve read names, but we cannot find a working mechanism in version 3 to remove them.) For a fair comparison, we stripped the read names before compressing. For Goby, we enabled the full "ACT H+T+D" compression scheme, as described in the Goby study [42].

### 2.3.1 Efficiency and compression ratio

We compressed each TopHat 2 alignment file with Boiler v1.0.1, CRAMTools v3.0, and Goby v2.3.5. Boiler was run with PyPy v2.4 and CRAMTools and Goby were run with Java v1.7. All tools were run on the Homewood High Performance

Compute Cluster at Johns Hopkins University. Each cluster computer has 2 Intel Xeon X5660 2.80GHz processors and 48 GB of RAM. We measured running time by adding the `user` and `sys` times reported by the Linux `time` command. Each tool runs predominantly on a single thread and processor. We measured peak memory usage in Python by spawning a new child process for the command and polling maximum resident set size (RSS) using the Python `resource` package's `getrusage` function. Peak memory usage for Boiler and Goby was consistent across runs, but CRAM memory usage varied widely between runs. We report the median peak memory of 10 runs for greater consistency.

Boiler takes roughly 1.5 – 2 times longer than CRAMTools and Goby to compress the D. melanogaster samples and about 2 – 12 times longer for the human samples (Table 2.2, Figure 2.2). It requires less memory than Goby for all but the deeper human samples, and less than CRAM for small datasets. For larger datasets, CRAM's memory footprint seems to be capped at around 2 GB (Table 2.3). Though Boiler allows the user to pose targeted queries without decompressing the entire file, we also evaluate how long Boiler takes to decompress an entire file relative to other tools. These results are presented in Table 2.3.1. Overall, Boiler takes roughly 2 – 4 times longer than CRAMTools and about 1 – 2 times longer than Goby to decompress entire files.

Importantly, Boiler achieves a compression ratio comparable to BigWig (Table 2.4) for all but the largest paired-end datasets, and usually produces far smaller com-

pressed files than CRAMTools or Goby. We measured both compressed file size (Table 2.4) and the "compression ratio" of original to compressed file size (Figure 2.3) for alignments generated by TopHat 2. The "original" file is a sorted BAM file with read names removed. For low-coverage unpaired datasets, CRAM and Goby's compression ratios are superior to Boiler's. However, we observe that while CRAMTools and Goby's compression ratios remain flat as the *D. melanogaster* library size increases, Boiler's ratios improve substantially (Figure 2.3), achieving its best compression ratios for the 20M-read samples: 56-fold for unpaired and 39-fold for paired-end samples. Boiler's compression ratio is consistently better than the other tools for paired-end samples and improves as library size increases. For high coverage *D. melanogaster* and all human datasets, Boiler achieves compression ratios 3-5-fold higher than both CRAM and Goby.

To demonstrate Boiler's performance on an established benchmark, we also ran Boiler on the dataset with Sample ID EJOYQAZ from the Goby study (Table 2.4). This consists of roughly 7.5 million paired-end reads from *H. sapiens*. The BAM file as well as the compressed Goby output is available at `data.campagnelab.org/home/compression-of-structured-high-throughput-sequencing-data`. Boiler produces a compressed file of 26.9 MB (2.8% of the original BAM) compared to Goby's output of 123.2 MB (13.0% of the original BAM), representing a 77% space reduction for Boiler compared to Goby. Boiler achieves a comparable compression ratio for alignments generated by HISAT (3.0% of the

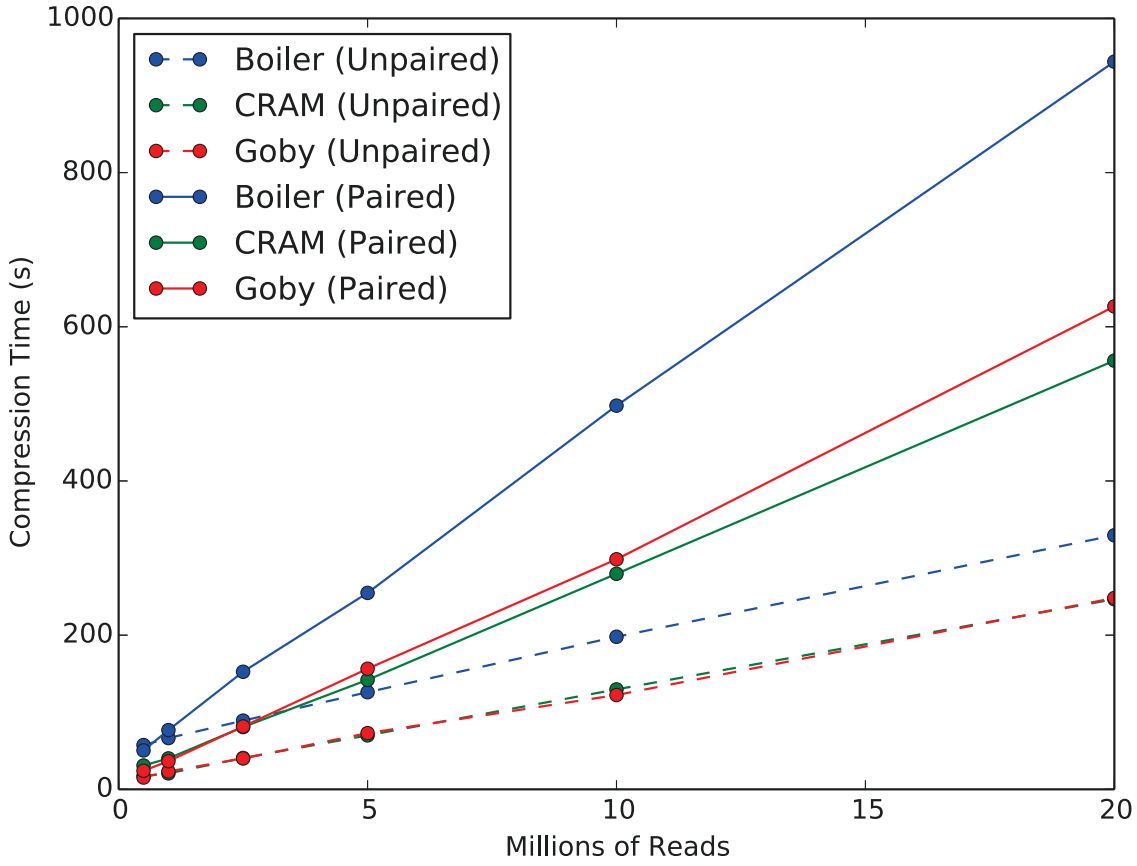


Figure 2.2: Time required for Boiler, CRAM and Goby to compress simulated *D. melanogaster* paired-end datasets.

original BAM).

### 2.3.2 Fidelity

Boiler discards read nucleotide and quality-value data. So while Boiler is not appropriate for pipelines where downstream tools measure non-reference alleles – e.g. for variant calling, allele-specific expression, or RNA editing – and may even limit post-hoc investigation of specific read alignments, we show that Boiler is appropriate

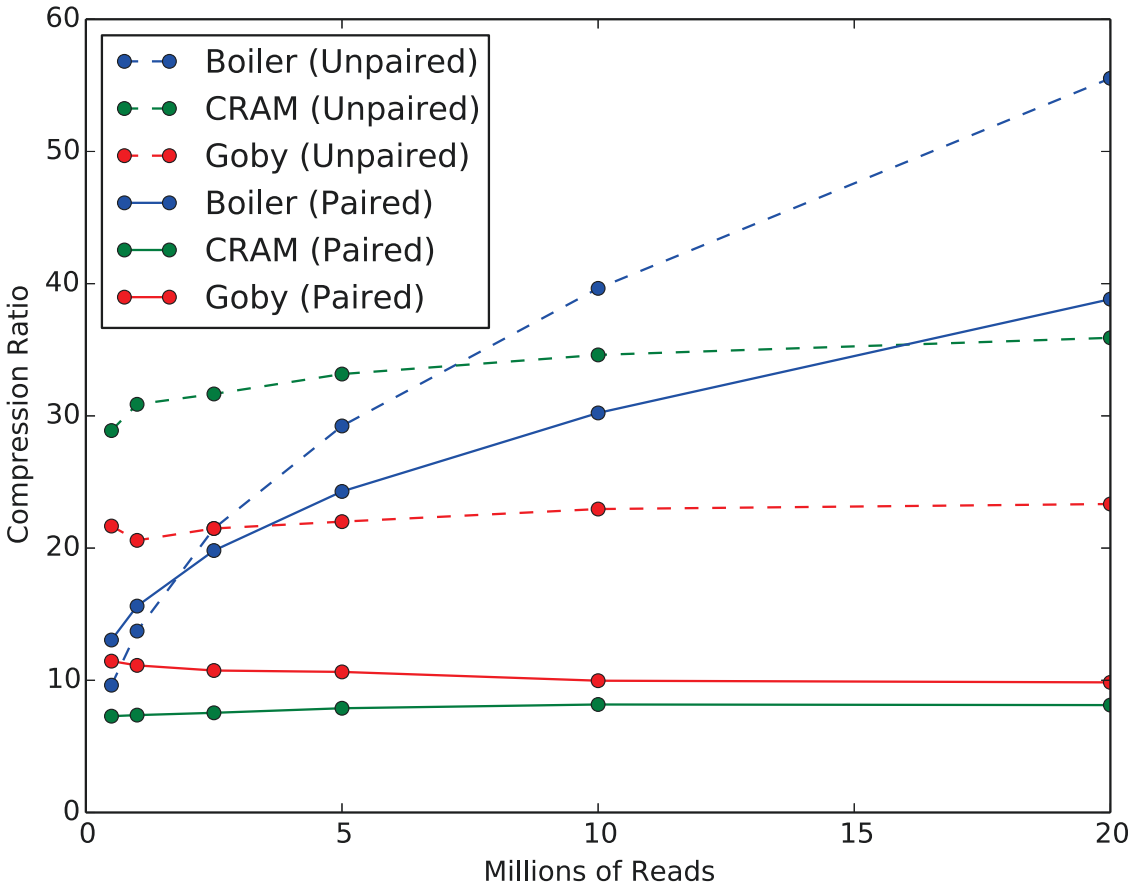


Figure 2.3: Compression ratios for simulated *D. melanogaster* datasets when compressed by Boiler, CRAM and Goby. Ratios are with respect to the original sorted BAM with read names removed.

for the common case where downstream tools are concerned with assembling and quantifying isoforms, e.g. Cufflinks and StringTie.

Boiler tends to "shuffle" alignment data in certain ways during compression. Figure 2.4 shows a simple case where Boiler might switch the order of two reads. Some of the shuffling is harmless, having no adverse effect on downstream results from Cufflinks and StringTie. But some shuffling could be harmful, negatively impacting the

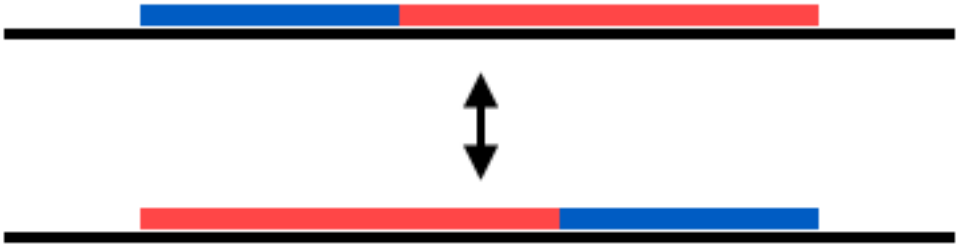

Figure 2.4: A simple example of two sets of read alignments which are indistinguishable by Boiler. Given just the coverage vector and read lengths, Boiler cannot determine the original read order.

fidelity of downstream results. Using both simulated and real data, we (a) establish the nature of the shuffling introduced by Boiler, (b) show there is only slight harmful shuffling in practice, and (c) show that the overall amount of shuffling is smaller – often much smaller – than the shuffling that results from substituting one technical replicate for another.

| Dataset | Boiler | CRAM | Goby |
|---|---|---|---|
| Drosophila, Simulated Unpaired | | | |
| 0.5M | 57.4 | 17.4 | 15.3 |
| 1M | 66.5 | 20.7 | 23.2 |
| 2.5M | 89.1 | 40.6 | 40.1 |
| 5M | 126.0 | 70.0 | 72.9 |
| 10M | 197.9 | 129.6 | 122.3 |
| 20M | 329.5 | 246.6 | 248.2 |
| | | | |
| Drosophila, Simulated Paired | | | |
| 0.5M | 50.5 | 30.9 | 24.1 |
| 1M | 76.8 | 40.1 | 36.5 |
| 2.5M | 152.5 | 80.8 | 81.5 |
| 5M | 254.8 | 142.0 | 156.4 |
| 10M | 497.7 | 279.6 | 298.4 |
| 20M | 943.9 | 556.1 | 626.4 |
| | | | |
| Human | | | |
| SRP025982 (11M) | 6122.6 | 507.6 | 674.5 |
| HG00100 (20M) | 4118.8 | 691.6 | 1045.6 |
| Simulated 20M | 2337.3 | 906.8 | 1301.3 |
| Simulated 40M | 7906.3 | 1476.8 | 2138.7 |

Table 2.2: Compression times in seconds.

| Dataset | Boiler | CRAM | Goby |
|---|---|---|---|
| Drosophila, Simulated Unpaired | | | |
| 0.5M | 0.38 | 0.74 | 0.73 |
| 1M | 0.29 | 1.37 | 1.39 |
| 2.5M | 0.24 | 1.12 | 1.72 |
| 5M | 0.24 | 1.13 | 1.73 |
| 10M | 0.24 | 0.96 | 2.11 |
| 20M | 0.50 | 1.53 | 3.77 |
| | | | |
| Drosophila, Simulated Paired | | | |
| 0.5M | 0.32 | 1.15 | 0.90 |
| 1M | 0.31 | 1.91 | 1.05 |
| 2.5M | 0.30 | 1.96 | 1.33 |
| 5M | 0.32 | 1.18 | 1.61 |
| 10M | 0.79 | 1.11 | 5.67 |
| 20M | 1.57 | 0.99 | 7.21 |
| | | | |
| Human | | | |
| SRP025982 (11M) | 7.10 | 2.08 | 5.3 |
| HG00100 (20M) | 3.23 | 2.08 | 4.12 |
| Simulated 20M | 6.11 | 2.08 | 4.87 |
| Simulated 40M | 9.14 | 2.08 | 6.88 |

Table 2.3: Peak memory usage (GB) reported by Python. Numbers reported are the median across 10 runs.

| Dataset | BAM | BigWig | Boiler | CRAM | Goby |
|---|---|---|---|---|---|
| Drosophila, Simulated Unpaired | | | | | |
| 0.5M | 26.0 | 3.7 | 2.7 | 0.9 | 1.2 |
| 1M | 49.4 | 6.4 | 3.6 | 1.6 | 2.4 |
| 2.5M | 120.3 | 12.8 | 5.6 | 3.8 | 5.6 |
| 5M | 222.2 | 19.0 | 7.6 | 6.7 | 10.1 |
| 10M | 436.1 | 28.1 | 11.0 | 12.6 | 19.0 |
| 20M | 883.0 | 37.0 | 15.0 | 23.2 | 35.7 |
| Drosophila, Simulated Paired | | | | | |
| 0.5M | 56.1 | 6.3 | 4.3 | 7.7 | 4.9 |
| 1M | 104.6 | 10.4 | 6.7 | 14.2 | 9.4 |
| 2.5M | 255.6 | 19.8 | 12.9 | 33.9 | 23.8 |
| 5M | 488.1 | 27.8 | 20.0 | 61.9 | 45.9 |
| 10M | 955.0 | 37.2 | 31.6 | 116.9 | 95.8 |
| 20M | 1902.6 | 47.8 | 49.0 | 226.2 | 193.3 |
| Human | | | | | |
| EJOYQAZ (7.5M) | 747.9[1] | – | 26.9 | – | 123.2[1] |
| SRP025982 (11M) | 810.1 | 54.6 | 38.0 | 160.5 | 159.8 |
| HG00100 (20M) | 2017.9 | 98.7 | 78.0 | 288.8 | 352.2 |
| Simulated 20M | 2117.3 | 54.3 | 71.6 | 273.5 | 307.8 |
| Simulated 40M | 3858.4 | 71.7 | 118.0 | 491.4 | 587.6 |

[1] Datasets downloaded from Goby supplementary data.

Table 2.4: Size of compressed files (MB) compared to the original sorted BAM with read names removed.

| Dataset | Boiler | CRAM | Goby |
|---|---|---|---|
| Drosophila, Simulated Unpaired | | | |
| 0.5M | 20.3 | 9.4 | 14.3 |
| 1M | 23.3 | 14.3 | 20.5 |
| 2.5M | 40.7 | 29.5 | 36.8 |
| 5M | 49.8 | 50.0 | 64.1 |
| 10M | 82.9 | 87.8 | 110.9 |
| 20M | 186.7 | 168.2 | 240.3 |
| | | | |
| Drosophila, Simulated Paired | | | |
| 0.5M | 31.7 | 16.1 | 23.0 |
| 1M | 44.5 | 25.7 | 35.8 |
| 2.5M | 97.5 | 55.4 | 70.8 |
| 5M | 161.0 | 99.0 | 121.6 |
| 10M | 344.5 | 192.6 | 239.2 |
| 20M | 1089.4 | 378.4 | 519.8 |
| | | | |
| Human | | | |
| SRP025982 (11M) | 828.2 | 315.1 | 654.0 |
| HG00100 (20M) | 1203.5 | 462.0 | –* |
| Simulated 20M | 1008.8 | 464.0 | 986.4 |
| Simulated 40M | 3179.7 | 835.3 | 1552.0 |

* Received an error when decompressing.

Table 2.5: Decompression times in seconds.

### 2.3.3 Alignment-level fidelity

Boiler compression can change where alignments lie on the genome and how they are paired. Here we ask how well alignment locations are preserved after Boiler compression of the TopHat 2-aligned samples. We measure alignment-level precision and recall in two ways. First we ignore read pairings. For each aligned unpaired read (or end of a paired-end read) in the original file, we seek a corresponding alignment in the compressed file where the genomic position of the alignment and of all overlapped splice junctions are identical. This counts as a true positive, and the alignments involved are "matched." An alignment can be matched with at most one other alignment. An alignment in the original file that fails to match an alignment in the compressed file counts as a false negative and the converse is a false positive Given these definitions, precision and recall are shown in the left-hand columns of Table 2.6 (labeled "Ignoring pairings"). These range from 96.2% to 99.4% across the samples tested.

We also measure precision and recall in a way that takes pairing into account: for each aligned pair, we seek a corresponding pair in the compressed file where both ends match their counterparts in terms of their genomic position and the positions of splice junctions. These results are shown in the right-hand columns of Table 2.6 (labeled "Including pairings"). Here precision and recall are lower, with most samples ranging from 17.6% to 54.5%. Note that the SEQC sample, SRP025982, exhibited higher unpaired precision and recall than the others (99.4%), and much higher than

the others when considering pairing (89.4%). This is likely due to the smaller number of reads in the sample relative to the other human samples, and to the much larger number of Boiler buckets induced by that sample. The larger number of buckets is likely owing to the presence of Universal Human Reference RNA, in which many genes are expressed.

We also measure genomic outer distance distribution (excluding unbundled alignments) before and after Boiler compression. We find they match closely (Figure 2.5, left) though not perfectly (Figure 2.5, right). The results are emblematic of Boiler's strategy: aggregate distributions are preserved, but links between particular alignments and particular points in the distribution are lost. As a result, some data is "shuffled;" ends themselves are largely unchanged, but pairings between ends are shuffled in a way that preserves the aggregate genomic outer distance distribution.

We repeated these experiments for alignments output by HISAT, as shown in Table 2.3.3. The results are similar to those produced by TopHat 2, with precision and recall ranging from 96.7% to 99.3% when ignoring pairings, and from 14% to 32.1% when considering pairings.

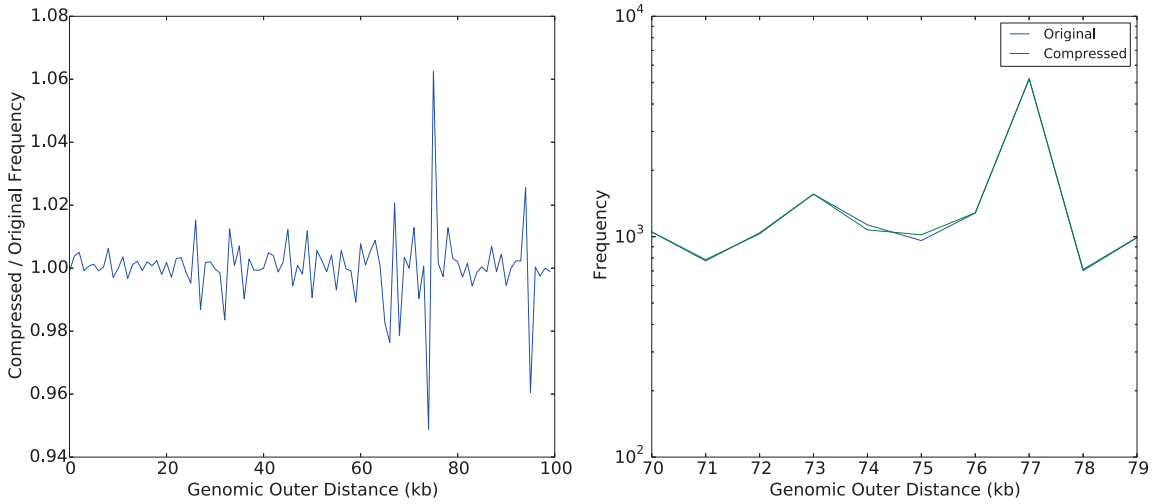


Figure 2.5: Comparison of genomic outer distances for the 10M paired-end *D. melanogaster* sample. **Left:** Frequency ratio of each genomics outer distance, compressed divided by original, up to a distance of 100,000 bases. **Right:** Original and compressed genomic outer distances between 70 and 79 kilobases in length.

| Dataset | Ignoring Pairings | | Including Pairings | |
|---|---|---|---|---|
| | Precision | Recall | Precision | Recall |
| Drosophila, Simulated Unpaired | | | | |
| 0.5M | 0.991 | 0.993 | – | – |
| 1M | 0.986 | 0.989 | – | – |
| 2.5M | 0.976 | 0.981 | – | – |
| 5M | 0.969 | 0.974 | – | – |
| 10M | 0.962 | 0.968 | – | – |
| 20M | 0.964 | 0.969 | – | – |
| Drosophila, Simulated Paired | | | | |
| 0.5M | 0.990 | 0.992 | 0.544 | 0.545 |
| 1M | 0.984 | 0.987 | 0.458 | 0.460 |
| 2.5M | 0.972 | 0.978 | 0.338 | 0.340 |
| 5M | 0.967 | 0.972 | 0.263 | 0.264 |
| 10M | 0.964 | 0.969 | 0.219 | 0.220 |
| 20M | 0.964 | 0.968 | 0.176 | 0.177 |
| Human | | | | |
| SRP025982 (11M) | 0.994 | 0.994 | 0.894 | 0.894 |
| HG00100 (20M) | 0.983 | 0.983 | 0.261 | 0.261 |
| Simulated 20M | 0.973 | 0.976 | 0.327 | 0.328 |
| Simulated 40M | 0.972 | 0.975 | 0.319 | 0.320 |

Table 2.6: Precision and recall of SAM reads aligned with Tophat.

| Dataset | Ignoring Pairings | | Including Pairings | |
|---|---|---|---|---|
| | Precision | Recall | Precision | Recall |
| Drosophila, Simulated Unpaired | | | | |
| 0.5M | 0.992 | 0.993 | – | – |
| 1M | 0.988 | 0.990 | – | – |
| 2.5M | 0.980 | 0.984 | – | – |
| 5M | 0.973 | 0.978 | – | – |
| 10M | 0.967 | 0.973 | – | – |
| 20M | 0.969 | 0.974 | – | – |
| Drosophila, Simulated Paired | | | | |
| 0.5M | 0.990 | 0.992 | 0.504 | 0.505 |
| 1M | 0.984 | 0.988 | 0.404 | 0.406 |
| 2.5M | 0.974 | 0.979 | 0.298 | 0.300 |
| 5M | 0.969 | 0.974 | 0.221 | 0.222 |
| 10M | 0.967 | 0.972 | 0.173 | 0.174 |
| 20M | 0.968 | 0.972 | 0.140 | 0.141 |
| Human | | | | |
| SRP025982 (11M) | 0.991 | 0.991 | 0.321 | 0.321 |
| Simulated 20M | 0.975 | 0.979 | 0.250 | 0.251 |
| Simulated 40M | 0.976 | 0.980 | 0.235 | 0.236 |

Table 2.7: Precision and recall of SAM reads aligned with HISAT.

### 2.3.4 Isoform fidelity

Having established Boiler's lossy-ness and shuffling behavior, we now assess the degree to which loss and shuffling have an adverse effect on downstream results obtained by Cufflinks v2.2.1 and StringTie v1.2.2. StringTie was run with default parameters. Cufflinks was run with the `--no-effective-length-correction` parameter to avoid variability due to an issue in how Cufflinks performs effective transcript length correction.

Let $T$ be the true simulated transcriptome, including abundances for each transcript, which we extract from the Flux-generated `.pro` and `.gtf` files. Let $\hat{T}$ be the transcriptome assembled and quantified from the original alignments, which we extract from the Cufflinks/StringTie output. Let $\hat{T}'$ be the same but for the Boiler-compressed alignments. Here we ask whether $\hat{T}$ and $\hat{T}'$ are approximately equidistant from $T$, indicating Boiler's loss and shuffling are not having an adverse effect.

We define a function for measuring the distance between two transcripts $t_1$ and $t_2$ assembled with respect to a reference genome. The function outputs a value between 0 and 1, with 0 indicating the transcripts do not match and 1 indicating a perfect match.

A transcript $t$ can be represented as a set of exons $\{e_1, ...e_n\}$, each defined by its start and stop positions. We first define a scoring function for two exons $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$:

$$s(e_1, e_2) = 1 - \frac{min(|x_2 - x_1|, k)}{2k} - \frac{min(|y_2 - y_1|, k)}{2k} \tag{2.1}$$

for some threshold $k$. We further define function $max(e, \hat{t})$ to be the set $\hat{E}$ of exons $\hat{e}$ from $\hat{t}$ with maximal score $s(e, \hat{e})$.

**Algorithm 1** Transcript scoring algorithm

```
1: procedure SCORE(t, t̂)
2:     s ← 0                                  ▷ Score before normalization
3:     n ← 0                                  ▷ Number of matching exons found
4:     for e ∈ t do
5:         Ê = max(e, t̂)
6:         for ê ∈ Ê do
7:             if e ∈ max(ê, t) then
8:                 s ← s + s(e, ê)
9:                 n ← n + 1
10:                break
11:            end if
12:        end for
13:    end for
14:    return s / (|t| + |t̂| − n)
15: end procedure
```

The weighted precision for pre-compression alignments is:

$$\sum_{\hat{t}\in\hat{T}} \max_{t\in T} \left(score(\hat{t},t)\right) \cdot c(\hat{t}) \tag{2.2}$$

Where $c(\hat{t})$ is the predicted coverage level of $t$ as reported by Cufflinks/StringTie. This measure is weighted both by the coverage of the assembled transcript and by the similarity of the matched-up transcripts. Precision for the post-compression alignments is calculated similarly, using $\hat{T}'$ instead of $\hat{T}$.

Similarly, weighted recall is

$$\sum_{t\in T} \max_{\hat{t}\in\hat{T}} \left(score(t,\hat{t})\right) \cdot c(t) \tag{2.3}$$

Where $c(t)$ is the true coverage level for $t$ as reported by Flux.

We investigated the effect of varying the threshold $k$ in the exon scoring equation (1). As $k$ increases, the scoring function becomes more relaxed, allowing exons with more divergent boundaries to contribute to the score. If $k = 0$, then two exons receive a score of 1 only if they are identical, 0 otherwise. On the other extreme, as $k \to \infty$ the score approaches 1 for all pairs of exons. Figure 2.6 shows the precision and recall at varying $k$ thresholds for the simulated *D. melanogaster* dataset containing 1 million paired-end reads. We observe that, across various $k$ cutoffs, accuracy decreases slightly after Boiler compression. Both plots show an inflection point around $k = 5$ to 10, after which the accuracy measure becomes more stable. Based on these results, we used a threshold of $k = 10$ in all experiments.

We calculated the weighted precision (Tables 2.8) and recall (Table 2.9) for all

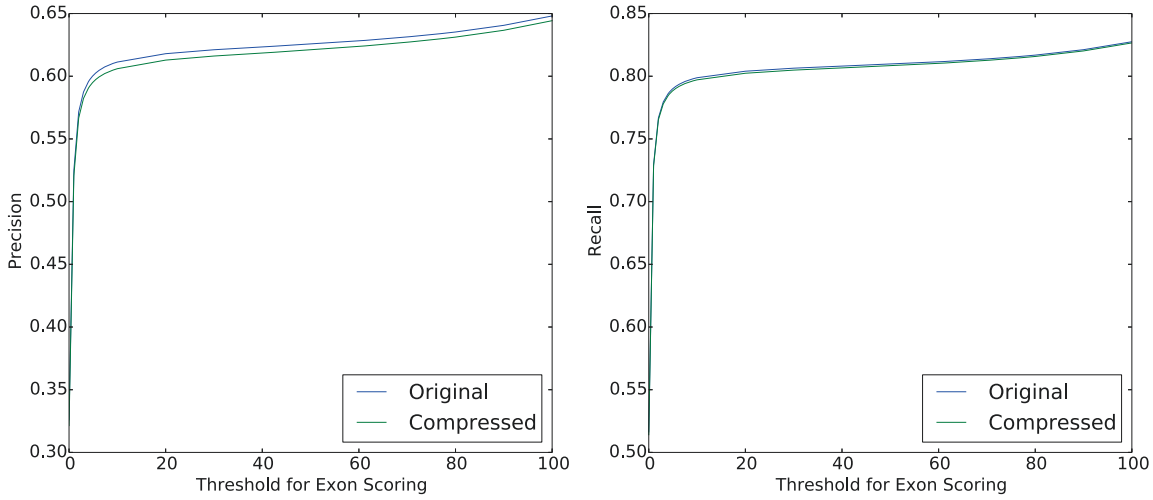


Figure 2.6: Precision (left) and recall (right) of transcripts compared to the reference transcriptome before and after compression with Boiler, as a function of the threshold used in the exon scoring function.

simulated samples. While there are small differences in all experiments (as expected due to Boiler's shuffling behavior) overall Boiler compression does not have a substantial adverse impact on weighted precision and recall. While precision and recall might not be uniform, and certain genomic regions may be disproportionately affected by compression, Boiler demonstrates consistent accuracy across both varying samples and varying scoring methods.

### 2.3.5 Other Isoform Fidelity Scores

In addition to precision and recall, we used two other methods to measure isoform accuracy, described below. *Weighted k-mer recall (WKR)* does not depend on a distance function, while the *Tripartite Score* compares two assembled transcriptomes

to a third "reference" transcriptome. The results from these methods, described in detail below, substantially agree with the results above.

**Weighted $k$-mer recall.**

We assess fidelity by measuring weighted $k$-mer recall (WKR), a component of the KC score developed by Li et al. [57] to assess transcriptome assemblies. WKR measures the degree to which an assembly recovers $k$-mers from the true simulated transcriptome, weighted by abundances of simulated transcripts containing the $k$-mer. For a $k$-mer $r$, its frequency profile $p(r)$ is defined as:

$$p(r) = \frac{\sum_{t \in T} n(r,t)c(t)}{\sum_{t \in T} n(t)c(t)}$$

where $T$ is the simulated transcriptome and for each transcript $t \in T$:

- $n(r,t)$ is the number of times $r$ occurs in $t$,
- $n(t)$ is the total number of $k$-mers in $t$, and
- $c(t)$ is the coverage of $t$.

Letting $R(T)$ be the set of all $k$-mers in transcriptome $T$:

$$WKR = \sum_{r \in R(T)} p(r)$$

WKR is defined with respect to the true transcriptome $T$, which we obtain from Flux Simulator's output. The GEUVADIS sample is not considered here, since it

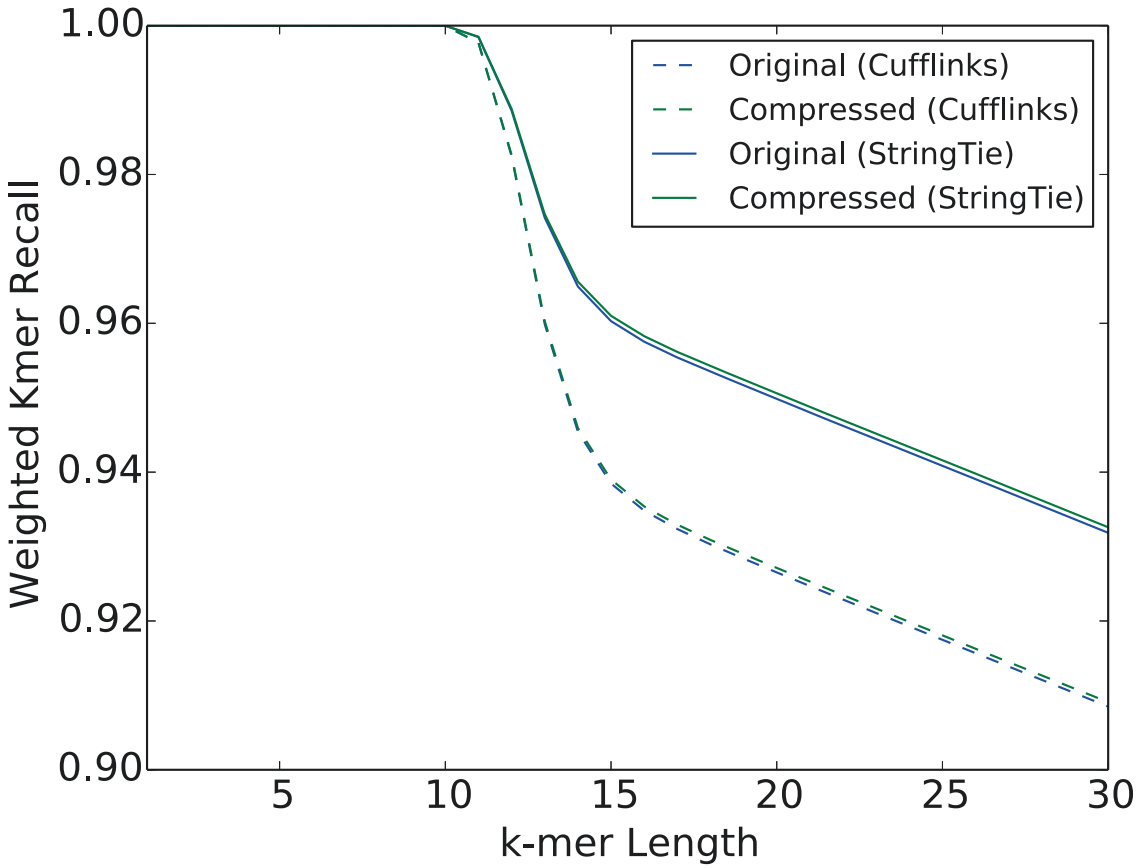


Figure 2.7: WKR with varying k-mer length for simulated Drosophila 10M paired-end reads, assembled with Cufflinks and StringTie.

is not simulated. Figure 2.7 shows that WKR is largely unchanged after Boiler compression for various $k$-mer length settings. It also shows that the difference in WKR is more pronounced for Cufflinks than for StringTie.

### Tripartite Score

We developed a different scoring method to compare the accuracy of alignments before and after compression, called the tripartite score. There are two versions of

this score, strict and loose.

We first construct a tripartite graph containing a node for each transcript in the cufflinks output for the alignments both before and after compression, as well as for each transcript in the reference transcriptome. We add a connecting edge from each transcript from the original set to the best-matching transcript from the reference set, determined using the transcript scoring method described previously. Similarly, we add an edge from each transcript in the compressed set to the best match from the reference set of transcripts.

For the strict tripartite score, we take all the nodes from the set of reference transcripts that are connected to a single node $A_i$ from the set of original transcripts and a single node $B_i$ from the set of compressed transcripts. The final score is the average of the transcript scores for every pair $A_i$, $B_i$.

For the loose tripartite score, we take all the nodes from the set of reference transcripts that are connected to at least one node from the set of original transcripts and at least one node from the set of compressed transcripts. Let $A_i$ be the original transcript with the highest score compared to the reference node, and let $B_i$ be the compressed transcript with the highest score compared to the reference transcript. The final score is the average of the transcript scores for every pair $A_i$, $B_i$.

Tables 2.10 and 2.11 show the tripartite scores alongside the percentage of transcripts from the original and compressed set of transcripts that contribute to the score.

### 2.3.6 Shuffling relative to technical replicates

Next we investigate the amount of "shuffling" introduced by Boiler – causing some reads to shift along the genome and scrambling some paired-end relationships – and whether the effect is large or small compared to the shuffling that occurs when switching from one technical replicate to another.

We first construct five artificial technical replicates by generating five times the desired number of reads with Flux Simulator and randomly partitioning the resulting read file five ways. We then assemble and quantify each using Cufflinks and StringTie. Let $\hat{T}_1$, $\hat{T}_2$, ..., $\hat{T}_5$ be the corresponding transcriptomes. We also pick a technical replicate ($\hat{T}_1$, say) to compress with Boiler. Let $\hat{T}_1'$ be the result of running Cufflinks/StringTie on the Boiler-compressed alignments. We calculate the weighted precision and recall of $\hat{T}_1'$ relative to $\hat{T}_1$. Finally, we calculate weighted precisions and recalls between all 10 ordered pairs of technical-replicate transcriptomes: $(\hat{T}_1, \hat{T}_2), (\hat{T}_1, \hat{T}_3), ..., (\hat{T}_4, \hat{T}_5)$. Since the ordering of technical replicates is arbitrary, the precision and recall values between these pairs are interchangeable. Results are presented in Tables 2.12 and 2.13. Precision and recall between technical replicates is shown as a range from the minimum to the maximum observed among the 10 ordered pairs. Precision and recall after Boiler-compression are consistently higher than precision and recall between technical replicates, indicating that shuffling due to Boiler compression is less consequential than shuffling due to technical variation.

For SRP025982, we followed the same process but using real lane-level technical

replicates. The same sample was sequenced in 5 separate lanes of an Illumina instrument, but on the same flowcell. We compressed the first replicate (SRR1216073) with Boiler and used the four others (SRR1216076 – SRR1216079) to calculate the ten pairwise precision and recall measures.

It is notable how precision and recall change relative to per-sample coverage. Precision and recall between technical replicates increases as per-sample coverage increases, indicating that the shuffling effect decreases as more transcripts become deeply covered. On the other hand, precision and recall of $\hat{T}_1'$ versus $\hat{T}_1$ decreases as per-sample coverage increases. Therefore, there may be higher levels of coverage for which Boiler shuffling has a greater impact than technical-replicate shuffling. For the realistic levels of coverage we tested, however, Boiler's shuffling remains less consequential.

### 2.3.7 Queries

Recovering coverage vectors from a Boiler-compressed file requires that Boiler decompress and combine coverage vectors for all the relevant bundles and buckets. Decompression of the coverage vector involves the DEFLATE algorithm and run-length decoding, but does not involve the more expensive read and pair recovery algorithms. SAMtools, on the other hand, does not explicitly represent the coverage vectors in a BAM file. Instead, coverage information must be recovered from the BAM file by first extracting the relevant alignments, then composing the coverage

vector using another tool like BEDTools:

```
samtools view -b -h x.bam c:start-end | genomeCoverageBed -bga -split
-ibam stdin -g chromosomes.txt
```

We compared the time required for Boiler to respond to coverage queries to the time required for SAMtools/BEDTools. Specifically, we iterated over all bundle boundaries in several *D. melanogaster* samples and queried for the coverage vector within those boundaries using both Boiler and SAMtools/BEDTools. Figure 2.8 compares the tools both in terms of average query time (left) and per-bundle query time (right). As expected, Boiler is consistently faster than SAMtools/BEDTools, with Boiler taking under 0.1 seconds on average, and SAMtools/BEDTools taking close to 0.75 seconds.

We also compared alignment query times for Boiler to those for the indexed BAM. A SAMtools alignment query uses this command:

```
samtools view -h x.bam chrom:start-end
```

Boiler must both decompress the relevant bundles and run the greedy read and pair recovery algorithms. Figure 2.9 compares the tools both in terms of average query time (left) and per-bundle query time (right). As expected, because of the need to run read and pair recovery, Boiler's alignment query is consistently slower than SAMtools. Even so, Boiler's average response time is under 0.15 seconds.

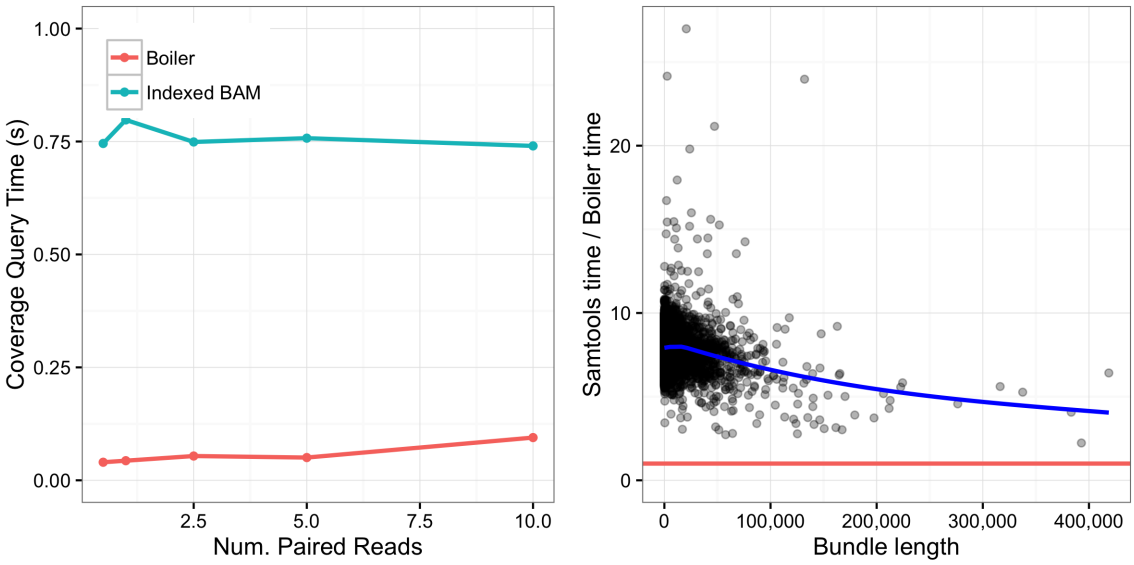


Figure 2.8: Comparison of coverage query times for Boiler the indexed BAM for all bundles. **Left:** Average query time for varying *D. melanogaster* paired-end datasets. **Right:** Ratio of Samtools / Boiler query time for each bundle in the 10M *D. melanogaster* paired-end dataset plotted as a function of bundle length. The blue line denotes the best fit line for the points, the red line is $y = 1$.

| Dataset | TopHat + Cufflinks | | TopHat + StringTie | | HISAT + StringTie | |
|---|---|---|---|---|---|---|
| | Original | Compressed | Original | Compressed | Original | Compressed |
| Drosophila, Simulated Unpaired | | | | | | |
| 0.5M | 0.364 | 0.364 (+0.1%) | 0.466 | 0.466 (+0.0%) | 0.542 | 0.542 (+0.0%) |
| 1M | 0.432 | 0.434 (+0.3%) | 0.545 | 0.545 (+0.0%) | 0.611 | 0.611 (+0.0%) |
| 2.5M | 0.527 | 0.527 (+0.1%) | 0.627 | 0.627 (+0.0%) | 0.646 | 0.542 (-0.7%) |
| 5M | 0.564 | 0.564 (+0.1%) | 0.637 | 0.637 (-0.0%) | 0.644 | 0.641 (-0.5%) |
| 10M | 0.583 | 0.585 (+0.2%) | 0.645 | 0.645 (+0.0%) | 0.639 | 0.637 (-0.3%) |
| 20M | 0.602 | 0.603 (+0.2%) | 0.654 | 0.653 (-0.1%) | 0.642 | 0.638 (-0.6%) |
| Drosophila, Simulated Paired | | | | | | |
| 0.5M | 0.583 | 0.582 (-0.1%) | 0.546 | 0.546 (-0.0%) | 0.628 | 0.628 (-0.0%) |
| 1M | 0.611 | 0.609 (-0.3%) | 0.614 | 0.613 (-0.1%) | 0.660 | 0.660 (-0.0%) |
| 2.5M | 0.632 | 0.630 (-0.4%) | 0.644 | 0.643 (-0.2%) | 0.666 | 0.665 (-0.2%) |
| 5M | 0.635 | 0.633 (-0.3%) | 0.652 | 0.652 (-0.1%) | 0.662 | 0.661 (-0.2%) |
| 10M | 0.644 | 0.643 (-0.1%) | 0.663 | 0.662 (-0.1%) | 0.666 | 0.665 (-0.1%) |
| 20M | 0.639 | 0.634 (-0.8%) | 0.660 | 0.659 (-0.2%) | 0.660 | 0.657 (-0.4%) |
| Human, Simulated Paired | | | | | | |
| 20M | 0.552 | 0.554 (+0.4%) | 0.570 | 0.568 (-0.4%) | 0.613 | 0.613 (-0.1%) |
| 40M | 0.554 | 0.555 (+0.2%) | 0.576 | 0.572 (-0.6%) | 0.614 | 0.615 (+0.1%) |

Table 2.8: Reference-based precision.

| Dataset | Cufflinks | | StringTie | | HISAT + StringTie | |
|---|---|---|---|---|---|---|
| | Original | Compressed | Original | Compressed | Original | Compressed |
| Drosophila, Simulated Unpaired | | | | | | |
| 0.5M | 0.583 | 0.583 (+0.0%) | 0.526 | 0.526 (+0.0%) | 0.551 | 0.551 (+0.0%) |
| 1M | 0.708 | 0.707 (-0.1%) | 0.682 | 0.682 (+0.0%) | 0.712 | 0.712 (+0.0%) |
| 2.5M | 0.791 | 0.785 (-0.7%) | 0.795 | 0.795 (-0.0%) | 0.803 | 0.800 (-0.4%) |
| 5M | 0.822 | 0.822 (+0.0%) | 0.834 | 0.834 (-0.0%) | 0.833 | 0.831 (-0.3%) |
| 10M | 0.824 | 0.824 (+0.0%) | 0.844 | 0.844 (+0.0%) | 0.835 | 0.834 (-0.2%) |
| 20M | 0.827 | 0.826 (-0.1%) | 0.851 | 0.850 (-0.1%) | 0.853 | 0.847 (-0.7%) |
| Drosophila, Simulated Paired | | | | | | |
| 0.5M | 0.732 | 0.729 (-0.4%) | 0.691 | 0.689 (-0.3%) | 0.715 | 0.715 (-0.1%) |
| 1M | 0.799 | 0.797 (-0.2%) | 0.792 | 0.791 (-0.1%) | 0.814 | 0.814 (-0.0%) |
| 2.5M | 0.825 | 0.826 (+0.1%) | 0.840 | 0.840 (-0.0%) | 0.843 | 0.840 (-0.4%) |
| 5M | 0.840 | 0.838 (-0.2%) | 0.857 | 0.855 (-0.2%) | 0.861 | 0.859 (-0.3%) |
| 10M | 0.828 | 0.824 (-0.5%) | 0.851 | 0.851 (-0.0%) | 0.851 | 0.849 (-0.3%) |
| 20M | 0.826 | 0.823 (-0.4%) | 0.850 | 0.849 (-0.1%) | 0.854 | 0.849 (-0.6%) |
| Human, Simulated Paired | | | | | | |
| 20M | 0.762 | 0.762 (+0.0%) | 0.794 | 0.792 (-0.2%) | 0.851 | 0.850 (-0.1%) |
| 40M | 0.792 | 0.789 (-0.3%) | 0.824 | 0.821 (-0.4%) | 0.851 | 0.849 (-0.2%) |

Table 2.9: Reference-based recall

| Dataset | Strict | | | Loose | | |
|---|---|---|---|---|---|---|
| | Score | % True | % Comp | Score | %True | % Comp |
| Drosophila, Simulated Unpaired | | | | | | |
| 0.5M | 0.999 | 27.5 | 27.6 | 0.981 | 43.7 | 43.8 |
| 1M | 0.999 | 23.3 | 23.4 | 0.986 | 39.4 | 39.6 |
| 2.5M | 0.998 | 24.9 | 24.9 | 0.992 | 34.7 | 34.7 |
| 5M | 0.996 | 23.9 | 24.0 | 0.994 | 30.6 | 30.7 |
| 10M | 0.992 | 23.1 | 23.1 | 0.990 | 28.0 | 28.1 |
| 20M | 0.988 | 22.3 | 22.4 | 0.986 | 26.9 | 26.9 |
| Drosophila, Simulated Paired | | | | | | |
| 0.5M | 0.994 | 45.2 | 45.2 | 0.984 | 59.2 | 59.2 |
| 1M | 0.990 | 35.5 | 35.5 | 0.983 | 47.1 | 47.1 |
| 2.5M | 0.979 | 31.5 | 31.3 | 0.977 | 39.4 | 39.2 |
| 5M | 0.979 | 28.0 | 27.9 | 0.976 | 34.4 | 34.4 |
| 10M | 0.969 | 25.2 | 25.2 | 0.963 | 31.6 | 31.6 |
| 20M | 0.965 | 23.3 | 23.3 | 0.959 | 29.5 | 29.5 |
| Human, Simulated | | | | | | |
| 20M | 0.976 | 16.8 | 16.9 | 0.972 | 22.6 | 22.7 |
| 40M | 0.976 | 14.4 | 14.5 | 0.970 | 20.2 | 20.3 |

Table 2.10: Tripartite score for Cufflinks transcripts.

| Dataset | Strict | | | Loose | | |
|---|---|---|---|---|---|---|
| | Score | % True | % Comp | Score | %True | % Comp |
| Drosophila, Simulated Unpaired | | | | | | |
| 0.5M | 1.000 | 43.2 | 43.2 | 1.000 | 59.1 | 59.1 |
| 1M | 1.000 | 44.0 | 4.0 | 1.000 | 58.3 | 58.3 |
| 2.5M | 0.999 | 40.9 | 40.9 | 0.997 | 51.1 | 51.1 |
| 5M | 0.998 | 34.3 | 34.3 | 0.998 | 42.3 | 42.3 |
| 10M | 0.993 | 29.4 | 29.4 | 0.994 | 36.4 | 36.4 |
| 20M | 0.988 | 24.5 | 24.5 | 0.990 | 32.0 | 32.0 |
| Drosophila, Simulated Paired | | | | | | |
| 0.5M | 1.000 | 44.6 | 44.7 | 0.999 | 58.5 | 58.6 |
| 1M | 0.999 | 41.0 | 41.0 | 0.994 | 51.9 | 52.0 |
| 2.5M | 0.996 | 35.1 | 35.0 | 0.996 | 42.7 | 42.7 |
| 5M | 0.995 | 28.5 | 28.6 | 0.994 | 36.2 | 36.2 |
| 10M | 0.989 | 25.0 | 25.0 | 0.988 | 32.5 | 32.6 |
| 20M | 0.986 | 23.0 | 23.1 | 0.986 | 30.3 | 30.3 |
| Human, Simulated | | | | | | |
| 20M | 0.985 | 16.7 | 16.6 | 0.985 | 22.4 | 22.4 |
| 40M | 0.983 | 14.0 | 14.0 | 0.981 | 19.9 | 19.9 |

Table 2.11: Tripartite score for Stringtie transcripts.

| Dataset | Cufflinks | | Stringtie | |
|---|---|---|---|---|
| | Boiler | Tech Reps (min–max) | Boiler | Tech Reps (min–max) |
| Drosophila, Simulated Unpaired | | | | |
| 0.5M | 0.997 | 0.386–0.396 | 0.998 | 0.452–0.470 |
| 1M | 0.996 | 0.492–0.502 | 0.998 | 0.559–0.578 |
| 2.5M | 0.989 | 0.659–0.669 | 0.993 | 0.729–0.736 |
| 5M | 0.988 | 0.757–0.763 | 0.989 | 0.807–0.816 |
| 10M | 0.983 | 0.814–0.824 | 0.985 | 0.854–0.863 |
| 20M | 0.980 | 0.861–0.868 | 0.983 | 0.892–0.900 |
| Drosophila, Simulated Paired | | | | |
| 0.5M | 0.986 | 0.618–0.636 | 1.000 | 0.568–0.583 |
| 1M | 0.980 | 0.724–0.733 | 0.996 | 0.698–0.706 |
| 2.5M | 0.965 | 0.806–0.811 | 0.989 | 0.807–0.815 |
| 5M | 0.969 | 0.843–0.850 | 0.991 | 0.854–0.862 |
| 10M | 0.960 | 0.867–0.879 | 0.986 | 0.890–0.895 |
| 20M | 0.952 | 0.893–0.902 | 0.985 | 0.915–0.921 |
| Human | | | | |
| SRP025982 (11M) | 0.964 | 0.627–0.645 | 0.997 | 0.622–0.641 |
| HG00100 (20M) | 0.939 | (no replicates) | 0.997 | (no replicates) |
| Simulated 20M | 0.969 | 0.897–0.912 | 0.981 | 0.910–0.932 |
| Simulated 40M | 0.965 | 0.911–0.927 | 0.976 | 0.935–0.947 |

Table 2.12: Non-reference-based precision. Columns labeled Boiler compare precision before and after Boiler compression. Columns labeled Tech Reps compare pairs of technical replicates.

| Dataset | Cufflinks | | Stringtie | |
|---|---|---|---|---|
| | Boiler | Tech Reps (min–max) | Boiler | Tech Reps (min–max) |
| Drosophila, Simulated Unpaired | | | | |
| 0.5M | 0.996 | 0.386–0.396 | 0.998 | 0.428–0.441 |
| 1M | 0.993 | 0.492–0.502 | 0.998 | 0.528–0.549 |
| 2.5M | 0.989 | 0.659–0.669 | 0.993 | 0.708–0.716 |
| 5M | 0.988 | 0.757–763 | 0.989 | 0.789–0.799 |
| 10M | 0.983 | 0.814–0.824 | 0.985 | 0.843–0.851 |
| 20M | 0.979 | 0.861–0.868 | 0.983 | 0.892–0.900 |
| Drosophila, Simulated Paired | | | | |
| 0.5M | 0.985 | 0.618–0.636 | 0.999 | 0.568–0.583 |
| 1M | 0.981 | 0.724–0.733 | 0.996 | 0.698–0.706 |
| 2.5M | 0.970 | 0.806–0.811 | 0.991 | 0.807–0.815 |
| 5M | 0.971 | 0.843–0.850 | 0.991 | 0.854–0.862 |
| 10M | 0.957 | 0.867–0.879 | 0.986 | 0.890–0.895 |
| 20M | 0.949 | 0.893–0.902 | 0.985 | 0.915–0.921 |
| Human | | | | |
| SRP025982 (11M) | 0.939 | 0.627–0.645 | 0.997 | 0.622–0.641 |
| HG00100 (20M) | 0.936 | (no replicates) | 0.995 | (no replicates) |
| Simulated 20M | 0.967 | 0.897–0.912 | 0.981 | 0.910–0.932 |
| Simulated 40M | 0.963 | 0.911–0.927 | 0.977 | 0.935–0.947 |

Table 2.13: Non-reference-based recall. Columns labeled Boiler compare recall before and after Boiler compression. Columns labeled Tech Reps compare pairs of technical replicates.

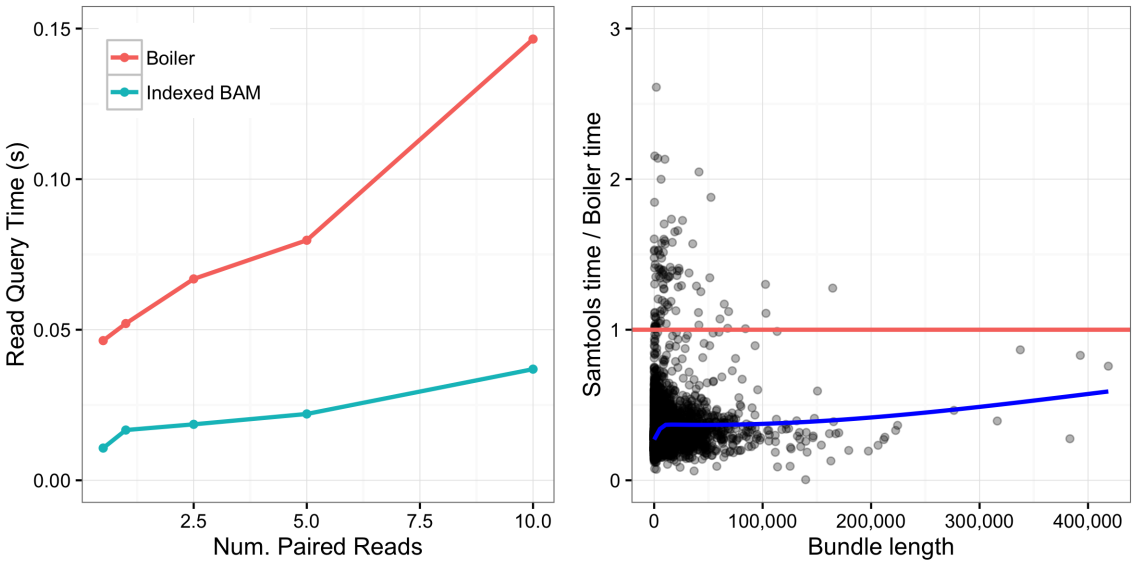


Figure 2.9: Comparison of alignment query times for Boiler versus sorted and indexed BAM for all bundles. **Left:** Average query time for varying *D. melanogaster* paired-end datasets. **Right:** Ratio of Samtools / Boiler query time for each bundle in the 10M *D. melanogaster* paired-end dataset plotted as a function of bundle length. The blue line denotes the best fit line for the points, the red line is $y = 1$.

## 2.4 Discussion

Boiler applies principles of lossy compression and transform coding to the problem of compressing RNA-seq alignments. Beyond discarding unnecessary BAM attributes, Boiler additionally discards most of the data that ties individual reads to their aligned positions and shapes. Boiler instead stores coverage vectors and read- and outer-distance tallies, effectively shifting from the "alignment domain" to the "coverage domain." While this can cause alignments to shift along the genome or pair with the wrong mate, the shuffling effect is modest compared to the differences between technical replicates, and adverse effects on downstream tools for isoform assembly and quantification are minimal.

Boiler is not a general-purpose substitute for RNA-seq SAM/BAM files, but it is an extremely space-efficient alternative that works well with tools like Cufflinks and StringTie. RNA-seq alignments are much larger than downstream files summarizing coverage (BigWig) or per-isoform expression level (FPKM table). Boiler and BigWig are similar in size, but, importantly, Boiler files preserve the ability to re-run the analysis repeatedly in the future. This can profoundly reduce the cost and difficulty of working with RNA-seq data, especially for large datasets.

Though we have not explored it here, we expect Boiler compression to work well with annotation-based tools like featureCounts [58] and HiTseq [59], as well as downstream differential-expression tools like derfinder [60] that rely on counts and coverage values.

Boiler also discards information about non-reference alleles, making Boiler archives readily sharable even when the input data is protected by privacy provisions like dbGaP.

In future work, it will be important to explore alternatives to the greedy decompression algorithms described here. For example, Boiler's greedy algorithm for extracting reads from a coverage distribution assumes the distribution of input read lengths has a dominant mode. This is a reasonable assumption for Illumina data, but not so for other sequencing technologies. Also, Boiler's compression and decompression algorithms could be accelerated by moving from Python to a compiled language such as C/C++. In general, there are many opportunities to make the read extraction and pairing algorithms more accurate, faster, and less hampered by assumptions about the data.

Another subject for future work is how Boiler represents multi-mapping alignments. By discarding read names, Boiler discards the one-to-many relationship between a multi-mapping read and its alignments. While this does not harm fidelity in most cases, it does adversely affect fidelity when Cufflinks is used to quantify from a gene annotation. It is an open question as to whether and how multi-mapping relationships can be represented in a way that allows trading off between compression ratio and fidelity.

Finally, we note that the methods used here to characterize Boiler's shuffling effect are more generally useful for evaluating any upstream tool that modifies the data. For

example, one could apply the same techniques to evaluate a tool for read trimming or digital normalization, or to compare many parameterizations of the spliced aligner.

Boiler is available from `github.com/jpritt/boiler` and is distributed under the open source MIT license.

# Chapter 3

# Optimal Graph Genome Construction with FORGe

## 3.1 Introduction

### Read alignment with variants

Read alignment is the process of determining each read's point of origin with respect to a reference genome. The origin can be ambiguous and reported alignments can be incorrect [61]. Repetitive genomes and sequencing errors contribute to this problem [61, 62]. Importantly, genetic differences between donor and reference genomes also contribute. Alignments overlapping positions where the genomes differ — i.e. where the donor genome has a non-reference allele — are systematically penalized. This can (a) reduce the correct alignment's score below the threshold

considered significant by the aligner, (b) cause the aligner's heuristics to miss the correct alignment, (c) cause the correct alignment's score to fall below the score at a different, incorrect location. The problem is magnified in hyper-variable regions such as the Major Histocompatibility Complex (MHC) [63, 64]. It is also problematic when individuals differ dramatically e.g. if they are from distinct inbred strains [20], or when downstream analyses are vulnerable to allelic bias, such as when detecting allele-specific expression [65, 21, 66] or calling heterozygous variants [67, 68].

Augmenting the reference genome with known variants helps in two major ways. First, it reduces the genetic distance between donor and reference genomes, removing the tendency to penalize correct alignments that overlap non-reference alleles. Second, it avoids the allelic bias, also called "reference bias," [65] that results when one donor haplotype resembles the reference more closely than the other(s).

There are many proposals for how to include and index genetic variants along with the reference genome. Two early approaches were GenomeMapper [20] and the Enhanced Reference Genome [21]. GenomeMapper came from a project to sequence many inbred strains of *Arabidopsis thaliana*, and it used a graph representation and an accompanying $k$-mer index to represent and align to a graph representing all strains. The Enhanced Reference Genome [21], which specifically addresses reference bias for allele-specific expression, included variants by taking the non-reference allele along with flanking bases and appending these "enhanced segments" to the linear reference genome. Since the resulting reference is linear, a typical read aligner like Bowtie [8]

can be used.

Several studies have expanded on these ideas. deBGA [69] uses a colored De Bruijn graph [70] and an accompanying hash-table index. BWBBLE [22] and gramtools [27] use an FM Index [71] with an expanded alphabet and modified backward-search algorithm to account for variants. GCSA [23] generalizes the compressed suffix array to index not a single reference but a multiple alignment of several references. HISAT2 [26] combines GCSA with the hierarchical FM Index implemented in HISAT [49]. GCSA2 [24] indexes paths in arbitrary graphs and is implemented in the VG software tool [25] which can align reads to such indexes. MuGI [72] and GraphTyper [68] use $k$-mer-based indexes.

Genome assemblies are also evolving along these lines. The GRCh37 and GRCh38 human assemblies [73, 74] include "alt loci," alternate assemblies of hypervariable regions including MHC. Other studies suggest modifying the linear genome by replacing each non-major allele with its major alternative [75, 76]. This leverages population-level information while keeping a linear representation.

**Variant selection and evaluation**

Past efforts that evaluated graph aligners have been selective about what variants to include in the graph, but without a clear rationale. Some included all variants from a defined subset of strains or haplotypes [20, 69, 72] or from a database such as the 1000 Genomes Project callset [6] or dbSNP [77]. In some cases, variants were filtered

according to ethnicity, e.g. keeping just the Finnish 1000 Genomes individuals [23] or the Yoruban HapMap [5] individuals [21]. The ERG study (concerned with allele-specific expression) excluded variants outside annotated genes. The gramtools study [27] used 1000 Genomes variants but excluded those with observed allele frequency less than 5%. GraphTyper [68] used dbSNP variants in one experiment, excluding single-nucleotide variants (SNVs) with under 1% frequency in all populations. HISAT2's software for selecting variants to include filters out SNVs with an allele frequency of under 10% in some cases [26].

Here we explicitly model the variants according to their effects on alignment, and we provide methods for choosing an optimal set based on those models. We apply these methods in combination with two different augmented-reference alignment methods, and compare to a range of relevant competing methods, including a linear reference with reference alleles, a linear reference with all-major alleles, and an ideal "personalized" reference that customized to fit the donor individual's alleles (including at heterozygous positions) as closely as possible. This experimental design allows us to make statements about how our methods affect accuracy, how those effects vary with genomic region, how close the methods come to achieving ideal accuracy, and how practical current graph alignment methods are overall.

## 3.2 Methods

FORGe works in cooperation with a variant-aware read aligner such as HISAT2 [26] or the ERG [21]. The strategy has two stages. In the *offline* stage, FORGe selects variants to include in the augmented reference based on a variant model — which predicts the pros and cons of including a variant — and a variant limit. The model and limit together constitute a *variant inclusion strategy* (VIS) that aims for a balance between accuracy and overhead. Once variants have been selected, the aligner software is used to create an index of the augmented reference. The second stage is an *online* stage where the read aligner aligns reads to the augmented reference using the index.

Inputs to the offline stage consist of (a) a reference genome, (b) variants in VCF format, (c) a VIS, and (d) a window size $s$. The variant inclusion strategy (VIS) consists of a variant model and a limit on the number or fraction of variants to include. The VIS is the user's most direct means for balancing blowup and alignment accuracy in the augmented reference. We now propose multiple variant models, each aiming to give higher scores to variants that will impart a greater net benefit when considering accuracy and blowup. The window size $s$ is used in three separate places in the software (described below) and should typically be set to the maximum read length.

### 3.2.1 Variant models

Let $G_{ref}$ denote the linear reference genome and $G^*$ the complete augmented genome including all variants in the population. Let $G$ be a possible result of a VIS, i.e. an augmented genome that includes a subset of population variants. For simplicity, assume all variants are SNVs (substitutions). Let a *localized s-mer* $\langle s, l\rangle$ be a string of length $s$ (the configurable windows size) that matches some combination of alleles in an augmented genome $G$ starting at offset $l$; we also call these simply $\langle s, l\rangle$-mers . For instance, if $G$ is `GATYACA`, where `Y` can be either `C` or `T`, then $\langle$`GAT`$, 0\rangle$, $\langle$`TCA`$, 2\rangle$ and $\langle$`TTA`$, 2\rangle$ are all $\langle 3, l\rangle$-mers of $G$. For an $\langle s, l\rangle$-mer $\sigma$, let $p(\sigma)$ be the probability a random $\langle s, l\rangle$-mer drawn from a random individual in the population equals $\sigma$. This can be calculated as:

$$p(\langle s, l\rangle) = p_l(l) \cdot p_s(\langle s, l\rangle) \approx \frac{p_s(\langle s, l\rangle)}{|G_{ref}|}$$

where $p_l(\sigma)$ is the probability a random $s$-mer begins at $\sigma$'s offset, which we approximate by $\frac{1}{|G_{ref}|}$. $p_s(\sigma)$ is the probability a localized $s$-mer starting at $l$ has alleles matching $\sigma$'s. We approximate $p_s(\sigma)$ by assuming independence and multiplying the frequencies of each allele, or, if phasing information is available, by using allele co-occurrence frequencies.

### Population Coverage

The *population coverage* $C(G)$ of an augmented reference $G$ is proportional to the population variation included, weighted by allele frequency. Specifically:

$$C(G) = \sum_{\langle s,l \rangle \in G} p(\langle s, l \rangle)$$

Note that $C(G_{ref}) \leq C(G) \leq C(G^*) = 1$.

We want to prioritize alleles according to how much they increase $C(G)$. To do so accurately, each variant's effect on $C(G)$ must be calculated according to which nearby variants (within $s - 1$ positions) are already in $G$. While this is possible, it requires much recalculation of scores as variants are added to $G$. It also means there is no way to produce a single, static list of per-variant model scores. For these reasons, we instead compute each variant's effect on $C(G)$ assuming that all surrounding variants are already in $G$; in other words, we compute the decrease in $C(G)$ caused by removing the variant from $G*$. We call this the *complete graph* assumption.

Although FORGe is capable of using phasing data — describing which alleles co-occur on the same haplotype — the complete graph assumption makes this irrelevant for our calculation here. We do make (optional) use of phasing data in the *Hybrid* model, discussed below.

### Uniqueness

The *uniqueness* $U(G)$ of a genome $G$ decreases as the multiplicities of its $k$-mers increase, i.e. as the genome becomes more repetitive (Figure 3.1). Uniqueness can be thought of as inverse to the amount of ambiguity, or repeated k-mers, in the genome. Let $f_G(s)$ be the number of $\langle s', l' \rangle$-mers in genome $G$ with $s = s'$. We define uniqueness of the genome as:

$$U(G) = \sum_{\langle s,l \rangle \in G} \frac{1}{f_G(s)}$$

Adding a variant to the genome can either increase or decrease $U(G)$. Specifically, an $\langle s, l \rangle$-mer overlapping the variant increases $U(G)$ if there is no other $\langle s', l' \rangle$-mers with $s = s'$. Alternately, an $\langle s, l \rangle$-mer overlapping the variant decreases $U(G)$ if there are any other $\langle s', l' \rangle$-mers with $s = s'$.

While we rely on this definition below, we do not expect uniqueness alone to be an effective variant model. This is because for most variants all the added (overlapping) $\langle s, l \rangle$-mers are unique. All such variants therefore receive an identical score. The hybrid measure, presented next, effectively breaks ties by also considering allele frequency.

### Hybrid score

The *hybrid score* $H(G)$ of a genome $G$ considers both population coverage and uniqueness. Again let $f_G(s)$ be the number of $\langle s', l' \rangle$-mers in $G$ with $s = s'$ and

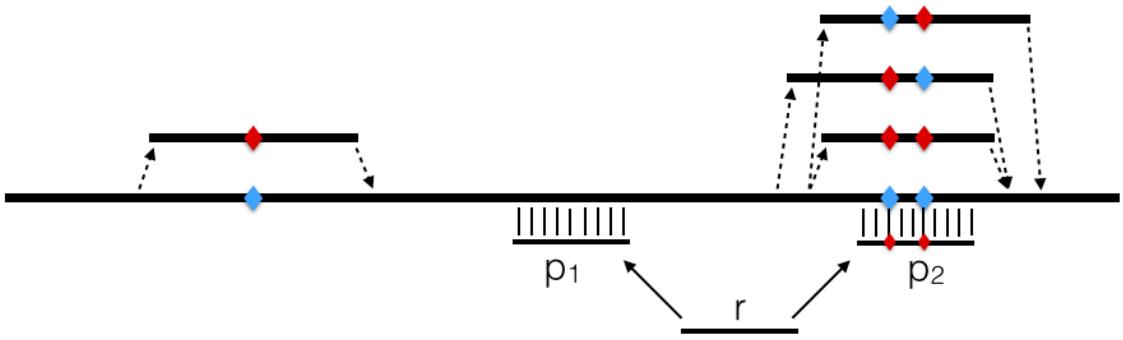


Figure 3.1: Simple example of added variants reducing uniqueness in the reference genome. Read $r$ maps uniquely to the linear reference genome at position $p_1$. The addition of the group of variants on the right creates a new perfect match for $r$ at position $p_2$, reducing overall uniqueness.

let $p(\langle s', l' \rangle)$ be the probability a random $\langle s, l \rangle$-mer drawn from a random individual equals $\langle s', l' \rangle$. We define the hybrid measure $H(G)$ of an augmented reference $G$ as

$$H(G) = \sum_{\langle s,l \rangle \in G} \frac{p(\langle s, l \rangle)}{f_G(s)}$$

Note that this is simply the dot product of the terms from the $C(G)$ and $U(G)$ sums.

For a variant $v$, we wish to compute the increase in $H(G)$ caused by adding $v$. For each $\langle s, l \rangle$-mer overlapping $v$ and containing the alternate allele, let $\langle s, l_1 \rangle, \langle s, l_2 \rangle, ..., \langle s, l_n \rangle$ be all other $\langle s, l \rangle$-mers with the same sequence $s$. Before adding $v$, the hybrid score can be written as

$$C + \sum_{i=1}^{n} \frac{p(\langle s, l_i \rangle)}{n}$$

where $C$ is the hybrid-score portion due to the $\langle s', l \rangle$-mers with $s' \neq s$. After adding

$\langle s, l\rangle$ to $G$, the score becomes

$$C + \sum_{i=1}^{n} \frac{p(\langle s, l_i\rangle)}{n+1} + \frac{p(\langle s, l\rangle)}{n+1}$$

The change in hybrid score due to the addition of $\langle s, l\rangle$ is

$$\Delta H_{s,l} = \sum_{i=1}^{n} \frac{p(\langle s, l_i\rangle)}{n+1} + \frac{p(\langle s, l\rangle)}{n+1} - \sum_{i=1}^{n} \frac{p(\langle s, l_i\rangle)}{n} = \frac{p(\langle s, l\rangle) - \frac{1}{n}\sum_{i=1}^{n} p(\langle s, l_i\rangle)}{n+1}$$

Assuming each $\langle s, l\rangle$-mer overlapping variant $v$ has a distinct sequence $s$, their $\Delta H_{s,l}$ terms are independent. Thus the total change in hybrid score due to the addition of $v$ is the sum of the $\Delta H_{s,l}$'s for each $\langle s, l\rangle$-mer overlapping and including $v$.

There are a couple caveats to how FORGe implements the hybrid model. First, As with the *Pop Cov* model, we make the complete graph assumption, allowing us to produce a scored variant list without dynamic re-scoring of variants as they are added. Second, computing $\Delta H_{s,l}$'s for all variants is expensive, since it involves calculating the read probability for each other occurrence of sequence $s$ for every overlapping $\langle s, l\rangle$-mer . Instead, we approximate it using average probabilities. Specifically, we pre-calculate $\bar{p}_{ref}$, the average $p(\langle s, l\rangle)$ for all $\langle s, l\rangle$-mers in $G_{ref}$, and $\bar{p}_*$, the average $p(\langle s, l\rangle)$ for all $\langle s, l\rangle$-mers in $G^*$ but not in $G_{ref}$. We approximate the summation with a weighted average:

$$\frac{1}{n}\sum_{i=1}^{n} p(\langle s, l_i\rangle) \approx \frac{1}{f_{G^*}(s)}[(f_{G^*}(s) - f_{G_{ref}}(s)) \cdot \bar{p}_* + f_{G_{ref}}(s) \cdot \bar{p}_{ref}]$$

Whereas the complete graph assumption rendered phasing data irrelevant to the *Pop Cov* model, we can use phasing data in the *Hybrid* model. This is because the *Hybrid* model weights the terms of the sum according to their frequency in the genome. By default, FORGe uses phasing information when it is available.

### Hybrid score implementation

The Uniqueness and Hybrid models are concerned with $s$-mer counts both in the linear reference genome ($G_{ref}$) and in the complete augmented reference ($G^*$). FORGe uses Jellyfish v2.2.6 [78] to calculate these counts. Since Jellyfish counts $s$-mers in a FASTA input file, FORGe must first construct an augmented FASTA such that $\langle s, l \rangle$-mers in this FASTA map one-to-one to $\langle s, l \rangle$-mers in $G^*$. This is also the goal of the Enhanced Reference Genome [21] representation, which accomplishes this by adding $2^k - 1$ "enhanced segments" for every length-$s$ window containing $k$ variants. Thus, to obtain $s$-mer counts for $G*$, we first constructed such a FASTA file using our implementation of the ERG, then counted $s$-mers using Jellyfish.

Once $s$-mers have been counted, FORGe computes the average probability for reads in the linear reference ($\bar{p}_{ref}$) and in the complete augmented reference ($\bar{p}_*$), for use in the *Hybrid* model formula. Finally, we compute the change in $H(G)$ for each $s$-mer in both and update the *Hybrid* model scores for every variant with an alternate allele in that read. After this, we have the full set of *Hybrid* model scores for all variants.

### Considering blowup

Adding variants to the augmented reference increases computational costs, including (a) size of the index on disk, (b) memory footprint during read alignment, and (c) time required for read alignment. We collectively refer to these as "blowup." Blowup is most drastic in genomic regions where variants are densely clustered, driving a exponential increase in the number of allelic combinations possible (Figure 3.2). A model based purely on minimizing blowup would prioritize isolated variants over those in clusters. We do not expect such a model to perform well on its own, though, since (like the *Uniqueness* model described above) it would fail to prioritize among the isolated variants. For this reason, we sought a way to combine a blowup avoidance strategy with the models already described above.

### Selecting variants with blowup avoidance

After ranking variants, FORGe selects the subset of variants to include in the augmented reference. The user specifies either a number of a fraction of all variants to include. In the simplest case, variants are chosen in order, starting with the highest-scoring variant, until the desired number have been included.

As an additional defense against blowup, we also propose a dynamic re-scoring scheme that can be added to an existing model. In this scheme, when a variant is added to the reference, FORGe searches for other variants within $s$ bases (the window length) of the added variant that have not yet been selected for addition.

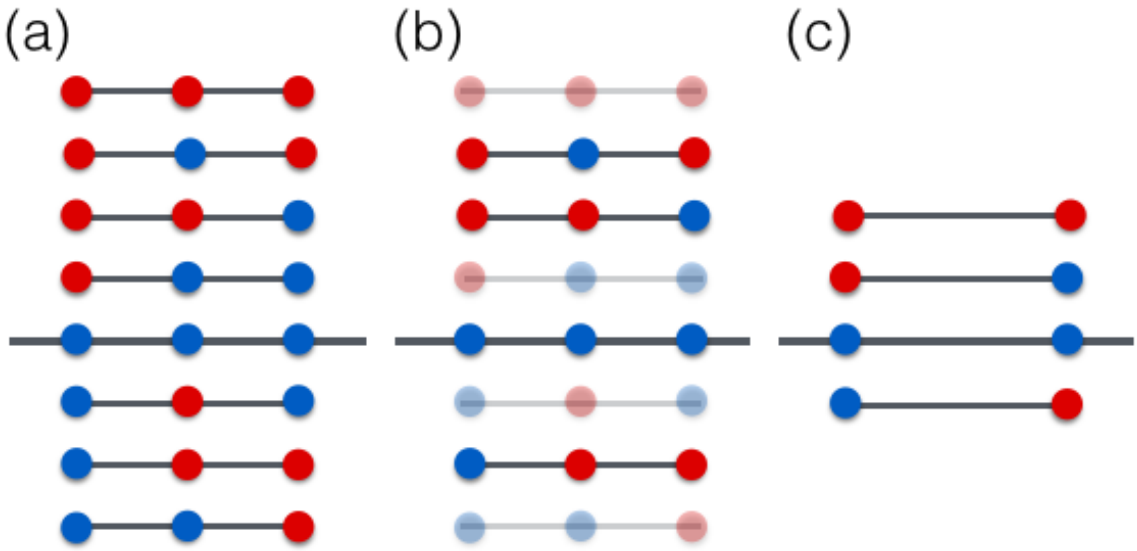


Figure 3.2: Demonstration of the blowup effect created by densely-packed variants. (a) In general, wherever $k$ variants lie inside our window, $2^k - 1$ new alternative contigs must be added to the graph. Current graph aligners often limit this blowup in one of two ways; either by (b) keeping only the most desirable alternative contigs (the method used by HISAT2's pruning script) or by (c) filtering the set of variants before creating alternative contigs (used by the ERG).

These nearby variants are re-scored by multiplying their score by a penalty factor $w$, where $0 < w \leq 1$. By letting $w$ be variable, FORGe can trade off between maximizing the model score and minimizing blowup. $w = 1$ maintains the original scores, whereas a penalty near $w = 0$ would ensure all isolated variants were added before any neighboring variants. We found that a penalty of $w = 0.5$ performed well in practice, and this is FORGe's default, used in all experiments performed here. *Pop Cov+* and *Hybrid+* are how we refer to those models when they are combined with this dynamic re-scoring scheme.

## 3.3 Results

### Strategy

Read alignment can be divided into *offline* (index building) and *online* (alignment) stages. FORGe operates in the offline stage. Specifically, FORGe takes a reference genome (FASTA format) and catalog of variants and their frequencies in the population (Variant Call Format). FORGe can also use phasing information when provided in the VCF. FORGe then selects variants to include in the index according to a model and desired fraction or number of variants.

### Simulation

We used Mason 0.1.2 [79] to simulate reads for testing. Mason simulates sequencing errors and base quality values. Mason also annotates each read with information about its true point of origin. We disabled Mason's facility for adding genetic variants, since we simulate from already-individualized references. We classify an alignment as correct if its aligned position in the reference is within 10 bases of the true point of origin. If the aligner reports several alignments for a read, we consider only the primary alignment — of which there is exactly one per aligned read, usually with alignment score equal to or greater than all the others — when determining correctness.

### Alignment

We tested FORGe with two read alignment strategies capable of including variants in the reference: HISAT2 [26] and the Enhanced Reference Genome (ERG) [21]. HISAT2 is a practical graph aligner that we hypothesized would benefit from careful selection of genetic variants to include. The ERG is simple and compatible with linear aligners like Bowtie. We use ERG only with short unpaired reads (25 nt) to test the hypothesis that the seed-finding step of an aligner can benefit from including FORGe-selected variants. While HISAT2 can be used with unpaired and paired-end reads, we test it only with unpaired reads here. Adapting the ERG approach to paired-end alignment is probably not practical (see Discussion).

In its offline stage, HISAT2 takes a linear reference genome and a VCF file with single-nucleotide variants and indels. HISAT2 uses GSCA indexing [23] to build a graph-genome index. The resulting graph is the generating graph for all combinations of reference (REF) and included alternate (ALT) alleles. HISAT2 also provides software that, starting from a VCF file (or the UCSC "Common SNPs" track, derived from dbSNP [77]), selects a subset of variants to include. It filters in two ways. First, it excludes variants with allele frequency under 10%. Second, where variants are densely packed, it imposes artificial haplotype constraints to avoid the exponential blowup that results from considering all combinations of REF and ALT alleles. We call this the *HISAT2 auto* method.

We also tested FORGe with our implementation of the ERG [21]. ERG's offline

phase starts with a linear reference genome and a variant file. It builds an augmented reference genome by adding *enhanced segments*: reference substrings that include ALTS and flanking context. The amount of context depends on a user-specified window size, $r$, which typically equals the maximum read length. When $n$ variants co-occur in a window, $2^n - 1$ enhanced segments are added to cover all combinations of ALT and REF alleles. The original ERG study limited growth by considering only the leftmost $k$ variants per length-$r$ window, with $k = 5$ in practice. We use a variation on this limit: if a window contains more than $k$ variants, we consider (a) the leftmost variant, and (b) the $k - 1$ other variants with highest allele frequency according to the input VCF. Including the leftmost guarantees that each variant has its ALT included in at least one of the overlapping enhanced segments. We also set the limit higher ($k = 15$) by default. While $k$ is configurable, we used the default in all experiments here. After adding enhanced segments to the reference, we indexed it with Bowtie [8]. In the online stage, we used Bowtie to align to the enhanced reference.

In all experiments, we ran HISAT2 with the `-k 10`, `--no-spliced-alignment`, and `--no-temp-splicesite` options. In the ERG experiments we ran Bowtie with the `-v 1` option to allow alignments with up to 1 mismatch. Note that HISAT2 is able to find alignments with mismatches, insertions or deletions, whereas Bowtie can only find alignments with mismatches. In all cases, we used Python's `rusage` module to measure peak resident memory usage and we used the Linux `time` utility to measure

running time. All tools were run using a single thread.

**Variant models**

As detailed in Methods, FORGe has two main models for ranking and selecting variants to include in the reference. First is *Population Coverage* (*Pop Cov*), which scores variants according to allele frequency. Second is *Hybrid*, which weighs both a variant's allele frequency and the degree to which its addition would make the reference more repetitive. Additionally, we evaluated versions of these two models enhanced with a *blowup avoidance* strategy that, at variant adding time, dynamically down-weights candidates that are close to already-added variants. These versions are called *Pop Cov+* and *Hybrid+*. All of these strategies are detailed in the Methods section.

### 3.3.1 Chromosome 9 simulation

We tested FORGe in a series of simulation experiments. We used human chromosome 9 from the GRCh37 assembly [73]. GRCh37 was chosen to match the coordinates for the official 1000 Genomes Project Phase-3 variants [6]. We simulated sequencing reads from chromosome 9 of NA12878, a female from the CEPH (Utah residents with Northern and Western European ancestry) group studied in the 1000 Genomes Project. Specifically, we generated 10 million unpaired Illumina-like reads from each haplotype of NA12878 for a total of 20 million reads. We created a VCF

file containing all single-nucleotide variants (SNVs) appearing in chromosome 9 in at least one 1000-Genomes individual, excluding NA12878 and family members. The resulting file contained 3.4 million SNVs. We used the *Pop Cov*, *Hybrid*, *Pop Cov+* and *Hybrid+* models to score the 3.4M SNVs. The *Hybrid* and *Hybrid+* models used phasing information, whereas the *Pop Cov* and *Pop Cov+* models did not (explained in Methods). We compiled subsets of SNVs consisting of the top-scoring 0%, 2%, 4%, 6%, 8%, 10%, 15%, and 20% up to 100% in 10 point increments.

**HISAT2**

Figure 3.3 shows alignment rate and accuracy when using HISAT2 to align our simulated 100bp reads to the genome indexes created with `hisat2-build`. The left-most point (in the case of 3.3c, the point labeled 0%) corresponds to a HISAT2 index with no SNVs added, i.e. a linear reference genome. The diamond labeled *Major Allele Ref* corresponds to a linear reference with all major alleles; i.e. with every SNV set to the allele that was most most frequent among CEU individuals in the filtered callset. The diamond labeled *HISAT2 auto* corresponds to the pruned set obtained by running HISAT2's scripts. The diamond labeled *Personalized* shows results when aligning to a personalized NA12878 genome with all non-reference homozygous (HOM) alleles replaced by their ALT versions and all heterozygous (HET) SNVs added as variants, so that neither REF nor ALT are penalized at alignment time. This is not a realistic scenario, but helpful for assessing how close the tested

methods come to the personalized-genome ideal. Plotted lines show results obtained when adding progressively larger subsets of SNVs to the graph genome, prioritized by model score.

Figures 3.3a and 3.3b show alignment rate and fraction of alignments that are correct (henceforth "correctness") as a function of the number of SNVs included in the genome. For all models except *Hybrid+*, peak alignment rate and correctness occur in the 8–12% range of SNVs included. All the FORGe models at their peak achieve higher alignment rate and correctness than the major-allele and HISAT2 methods. When greater fractions of variants are included — more than around 12% — alignment rate and correctness generally decrease. Correctness eventually decreases to a level only somewhat higher than that achieved by the linear reference, showing that alignment suffers when too many variants are included. Figures 3.3d and 3.3e are similar to 3.3a and 3.3b but show alignment rate and correctness as a function of HISAT2's memory footprint at alignment time. While FORGe's models at their peak have a roughly 50% larger memory footprint than the linear references (both major-allele and reference-allele), they use roughly half the memory of the "HISAT2 auto" method.

Figure 3.3c plots a point or a parametric curve for each indexing strategy and model. The vertical axis is the fraction of reads (not alignments) that aligned correctly, and the horizontal axis is the fraction of reads that aligned incorrectly. Notable points on the curves are labeled with the fraction of SNVs included. Diamonds mark

points on the curves with maximal $y - x$, where $y$ is fraction correct and $x$ is fraction incorrect. This is a combined measure for alignment rate and accuracy, and maximal values are reached in the 8–10% range of SNVs included (except *Hybrid+*, which peaked at 30%). The best-performing are superior to (above and to the left of) the linear-genome methods, the "HISAT2 auto" method, and to the the genome obtained by adding all of the SNVs (labeled 100%). The best-performing graph genomes come much closer to the personalized-genome ideal than the other methods.

It is notable that the alignment rate curves in Figure 3.3a,b,d and e eventually trend downward. Like most read aligners, HISAT2 uses heuristics to limit the effort spent aligning reads to many repetitive regions of the same reference genome. HISAT2 is unusual in that when a read has too many repetitive alignments, it will abort and leave the read unaligned. Bowtie does not have this heuristic; rather, Bowtie chooses one best-scoring alignment to report even when the read has many repetitive alignments. Because of this, HISAT2's alignment rate decreases as more variants are included and the genome becomes more repetitive.

A known drawback of graph aligners is that accuracy and overhead can suffer when many variants co-occur in a small window of the genome. To measure the impact this has on FORGe's models, we also plotted results using *blowup avoiding* versions of the *Pop Cov* and *Hybrid* models (Figure 3.3, dotted lines), called *Pop Cov+* and *Hybrid+*. These versions will, when selecting variants to add, deprioritize variants that are near already-added variants. We observed that blowup avoidance

had a minimal impact on the shape of the *Pop Cov* curve; e.g. Figure 3.3d & e shows the solid and dotted lines for *Pop Cov* on top of each other. Notably, blowup avoidance did cause the alignment memory to increase more slowly with respect to the number of added variants for the *Pop Cov* ranking (Figure 3.3f). For the *Hybrid* model, blowup avoidance did not change the relationship between memory footprint and number of variants added (Figure 3.3f) and had an adverse effect on alignment rate and correctness. This is likely because the *Hybrid* model already takes clustered variants into account in its $k$-mer counts.

We repeated these experiments for paired-end reads (Figure 3.4) and the results closely followed those in Figure 3.3. As expected, alignment rate and accuracy both increase when using paired-end reads, since an accurate alignment for one end can "rescue" the other in the presence of ambiguity. Peak accuracy (maximal $y - x$) was achieved at the same SNV fraction except in the case of the Hybrid ranking, which peaked at 15% rather than at 10%.

We also repeated these experiments for reads simulated from a YRI individual (Figure 3.5), taking the same measures we took above to exclude the individual and family members before providing variants to the model for scoring. These results also closely follow those in Figure 3.3, with accuracy and recall peaking at a somewhat higher percentage of variants included (15% for YRI compared to 8-10% for CEU), likely due to YRI's greater divergence from the reference. We return to this in the Discussion.

### Enhanced Reference Genome

Figure 3.6 shows alignment rate and correctness when using Bowtie [8] to align simulated 25bp reads to enhanced references constructed with the ERG method [21]. We used shorter reads and configured Bowtie to find alignments with up to 1 mismatch (`-v 1`) to mimic the seed alignment step of seed-and-extend aligners.

Unlike HISAT2, Bowtie always reports an alignment if one is found, regardless of how repetitively the read aligns. Consequently, the alignment rate shown in Figure 3.6a and d strictly increases as variants are added to the graph. Apart from that, the results reinforce those from Figure 3.3. Peak alignment rate occurs at a relatively small fraction of SNVs (6-20%). As more variants are added, decreases eventually decreases, though the *Hybrid* ranking does not suffer this drop until over 70% of SNVs are included. The alignment-time memory footprint of the best-performing FORGe indexes is higher than that of the linear reference; e.g., including the top 6% of *Pop Cov+*-scored SNVs increases the footprint 29%, from 127.9 MB to 165.0 MB. But it is a fraction of the size of the index when 100% of variants are included (1.87 GB). Blowup avoidance (Figure 3.6, dotted lines) had a somewhat minor effect on alignment rate and correctness for *Pop Cov*, and a clear negative effect for *Hybrid*. On the other hand, it slowed the rate of index growth for both models at low and intermediate fractions of SNVs (Figure 3.6f).

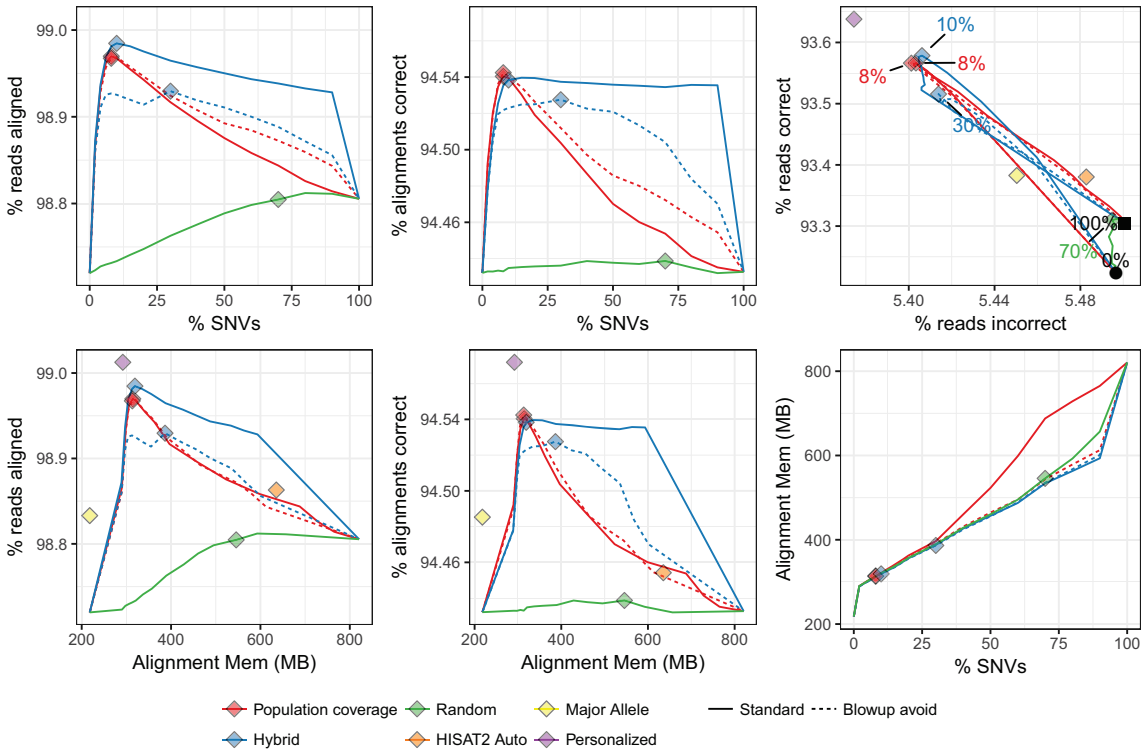


Figure 3.3: Results from NA12878 simulation. 100-bp unpaired reads were simulated from GRCh37 Chromosome 9 with NA12878's variants included. FORGe and HISAT2 created and indexed augmented reference genomes with various variant sets. Besides the *Pop Cov* and *Hybrid* rankings, we also included a strategy that gave variants random ranks ("Random"). (a) and (d) show the fraction of reads aligned. (b) and (e) show the fraction that aligned correctly to the simulated point of origin. (c) plots a parametric curve of the fraction of reads with a correct alignment (vertical) versus the fraction with an incorrect alignment (horizontal). Lines follow measurements made over a range of fractions of SNVs, with points for 0%, 2%, 4%, 6%, 8%, 10%, 15%, and 20% up to 100% in 10 point increments.The diamond labeled *HISAT2 auto* is an augmented genome produced using HISAT2's pruning scripts. The diamond labeled *Major allele ref* is a linear reference with all positions set to the most frequent allele. Other diamonds indicate the SNV fraction maximizing $y - x$, where $y$ is the fraction of reads aligned correctly and $x$ is the fraction aligned incorrectly. The *HISAT2* and *Major allele* diamonds are excluded from panels (a), (b) and (f) because there is no clear way to measure the fraction of variants included by these methods. The black filled circle and square in panel (c) represent measurements when 0% and 100% of variants are included, respectively.

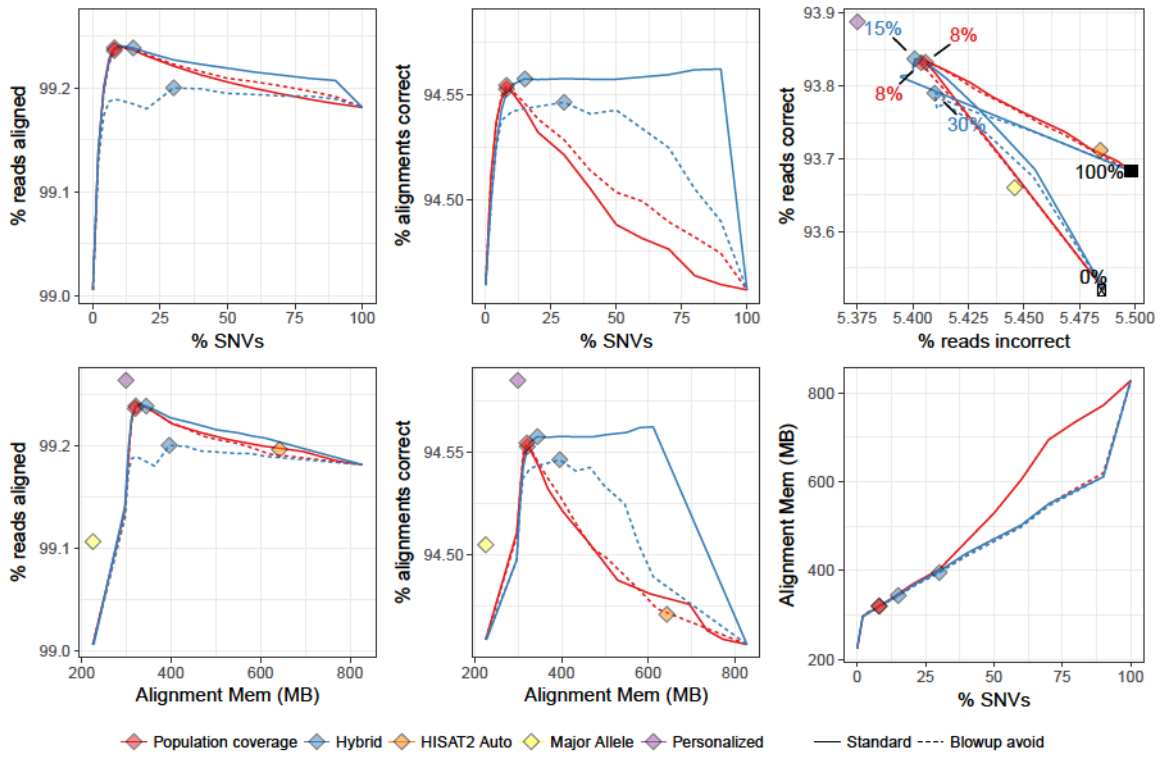


Figure 3.4: Results from aligning simulated 100-bp paired-end reads to GRCh37 chromosome 9 using HISAT2. Panels and plots have the same meaning as for Figure 3.3 except that the random ranking is omitted here.

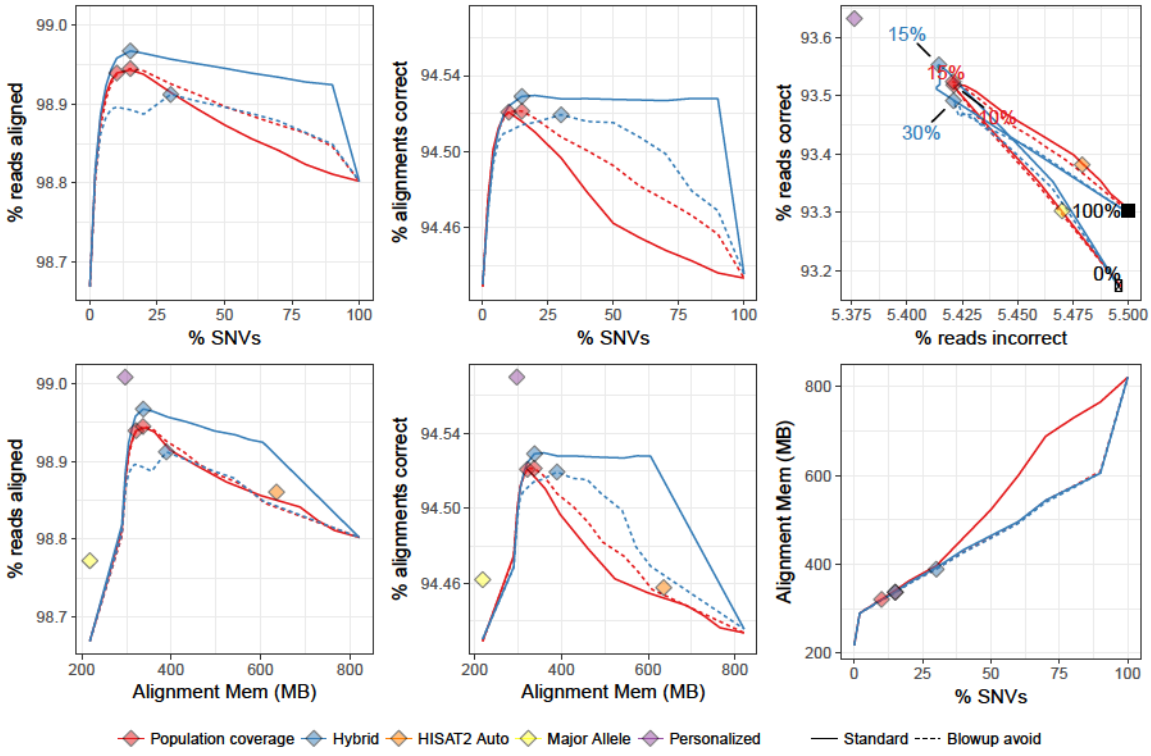


Figure 3.5: Accuracy for reads simulated from chromosome 9 of YRI individual NA19238. The initial variant set contained all 1000 Genomes SNPs in chromosome 9, after removing NA19238 and its direct relatives. Panels and plots have the same meaning as for Figure 1 except that the random ranking is omitted here.

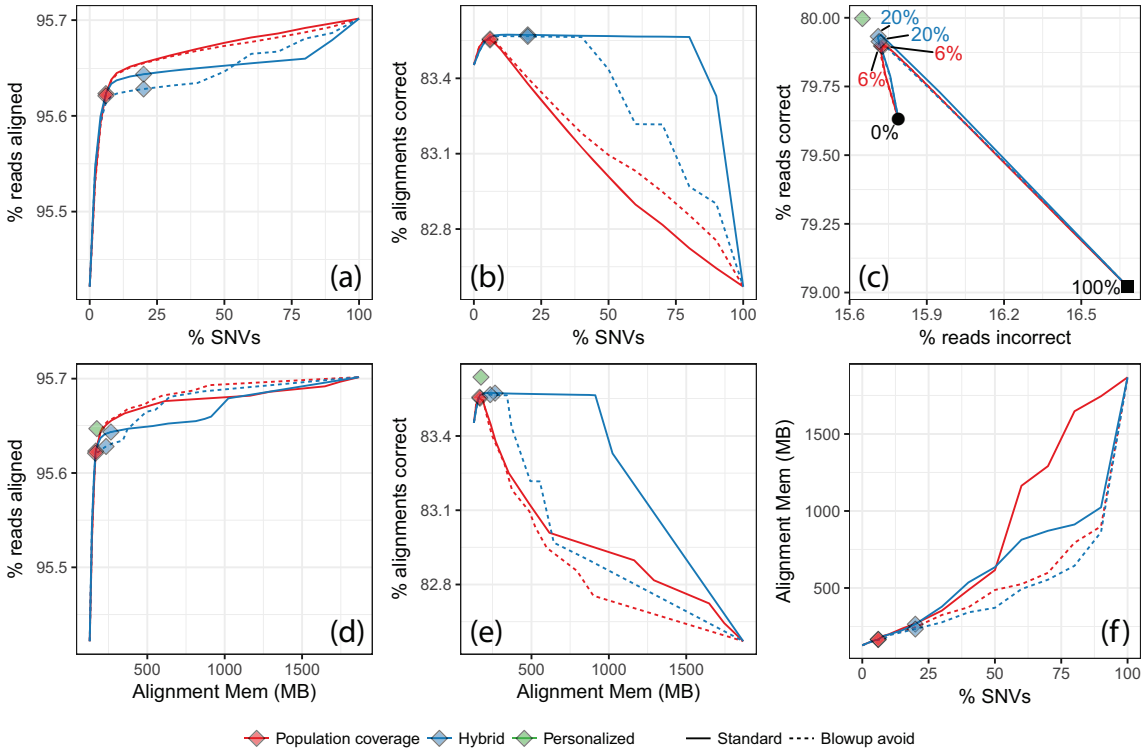


Figure 3.6: Results from aligning simulated 25-bp unpaired reads to GRCh37 chromosome 9 using the ERG+Bowtie approach. Panels and plots have the same meaning as for Figure 3.3 except that *HISAT2 auto* and *Major allele ref* diamonds are omitted here.

**Stratification by variant density, variant rarity, and repetitiveness**

Figure 3.3c showed that when we move from 0% to 8% of variants included in the augmented reference, the number of correct alignments increases by about 0.4 percentage points (as a fraction of reads) and the number of incorrect decreases by about 0.1 points. Though these may seem like small differences, in a study with 1.2 billion reads — approximately the number of unpaired 100 nt unpaired reads required to cover the human genome to 40-fold average depth — this would yield about 4.8M more correctly aligned reads and 1.2M fewer incorrectly aligned.

Still, we hypothesized that certain read subsets might be affected more dramatically by the inclusion of variants. To this end, we measured alignment rate and correctness when we varied the number of alternate alleles overlapped by a read (3.7a-c), whether the alternate allele was common or rare (3.7d-f) and what kind of genomic region or repeat the read originated from (3.7g-i). The measurements studied here are the same as those presented in Figure 3.3, but filtered as described below.

Figures 3.7a-c show alignment rate and correctness stratified by the number of non-reference SNVs overlapped by a read. To obtain these subsets, we first removed reads originating from reference-genome regions deemed repetitive by DangerTrack [80] (score over 250). We did this after finding that these regions had a combination of low SNV density and repetitive content that caused the 0-SNV stratum to behave very differently from the others. Reads containing 1 or more SNVs undergo a rapid increase in alignment rate and correctness from 0% to 10% of SNVs. Beyond 10%, all strata

experience a slow decrease in alignment rate and correctness up to 100% of SNVs added. The 0-SNV stratum has decreasing alignment rate and correctness across the whole range, as expected since the addition of variants cannot help (since the reads lack alternate alleles) but can harm alignment by increasing the repetitiveness of the reference. Strata with more SNVs experience a more dramatic rising-and-falling pattern; for the 3-SNV stratum, alignment rate varies from about 80–98%. While curves for the various strata have different shapes, all peak at a relatively low SNV fraction: 20% or lower.

Figures 3.7d-f show alignment rate and correctness for reads containing a single rare SNV allele (1000 Genomes frequency $< 0.5$) versus reads containing a single common SNV allele ($\geq 0.5$). In both cases, we considered only reads with a single non-reference allele. Rare-SNV reads peak lower and at a higher SNV fraction than common-SNV reads for both alignment rate and correctness (Figures 3.7d-f). This is expected, since the *Pop cov* model prioritizes common over rare SNVs. In other words, by the time a rare variant is added, many common variants have already been added, making the genome more repetitive.

Figures 3.7h-j show alignment rate and correctness for reads stratified by feature of origin. We analyzed reads originating from (a) RepeatMasker-annotated repetitive regions (`http://www.repeatmasker.org`), (b) RepeatMasker-annotated "Alu" repeats, (c) regions captured by the Nextera exome sequencing protocol, and (d) all reads. Reads from repetitive regions generally had lower alignment rate and correct-

ness compared to all reads. As before, alignment rate and correctness curves peaked at low SNV fractions: 10% or lower. Reads from more repetitive features were more sensitive to the number of variants included in the reference, as evidenced by the vertical spans of the curves.

In a related experiment, we examined the graph genome's effect specifically on the hypervariable MHC region. We simulated reads from NA12878 Chromosome 6 and used HISAT2 to align to both a linear and a graph genome augmented with the top-scoring 10% of SNVs. We visualized the read-alignment pileup in the hypervariable MHC region using IGV [81] (Figure 3.8). Qualitatively, the pileup for the augmented reference looks superior — with more coverage in variant-dense regions and with more even overall coverage — to the pileup for the linear reference.

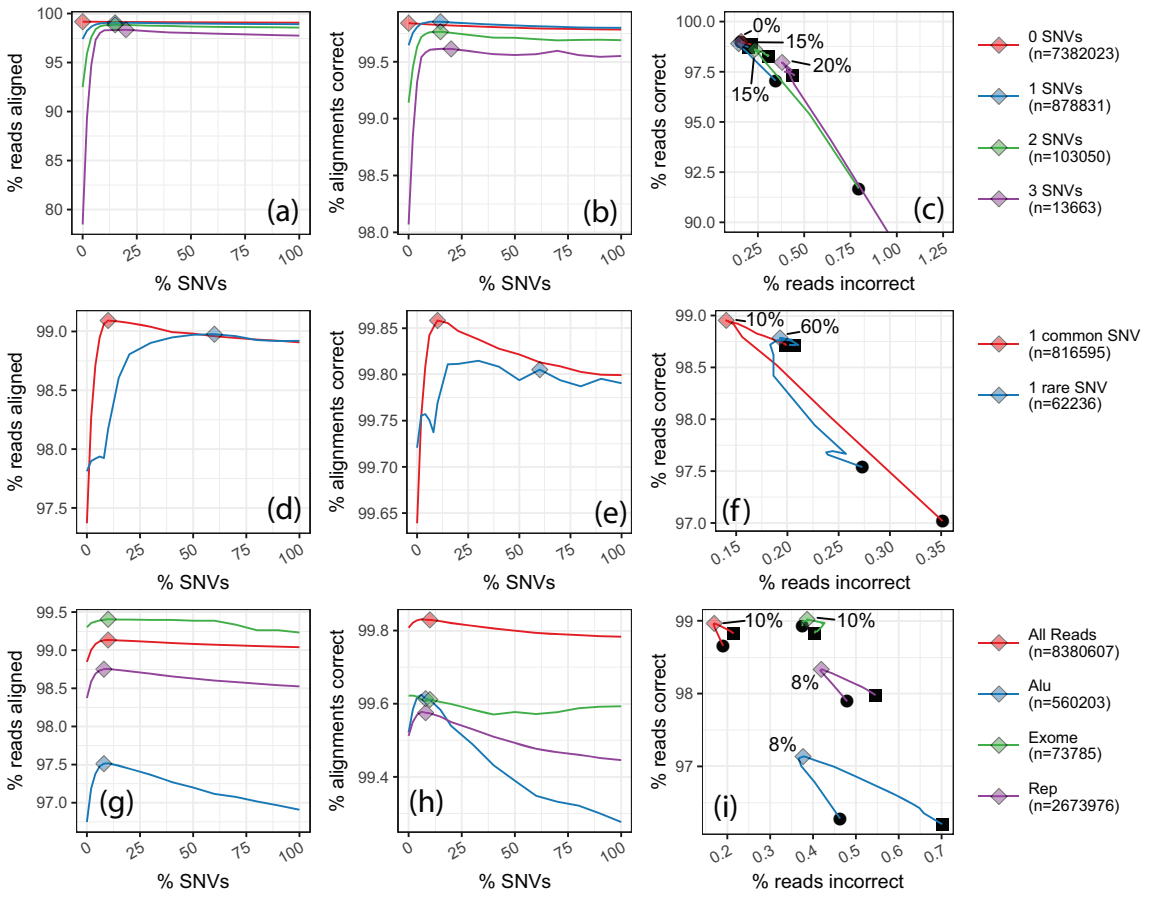


Figure 3.7: First row: Results for simulated reads stratified by the number of SNV alternate alternate alleles overlapped by the read. Reads overlapping regions with high DangerTrack [80] score — indicating the regions are difficult to align to — are omitted. Second row: Results for simulated reads overlapping exactly one common alternate allele (and no other alternate alleles) and reads overlapping exactly one rare allele. Reads overlapping regions with high DangerTrack [80] score, indicating the regions are difficult to align to. Third row: Results for simulated reads stratified by region of origin. Regions examined are: regions labeled with the "Alu" family by RepeatMasker, regions captured by the Nextera exome sequencing protocol ("Exome"), and regions labeled with any repeat family by RepeatMasker ("Rep").

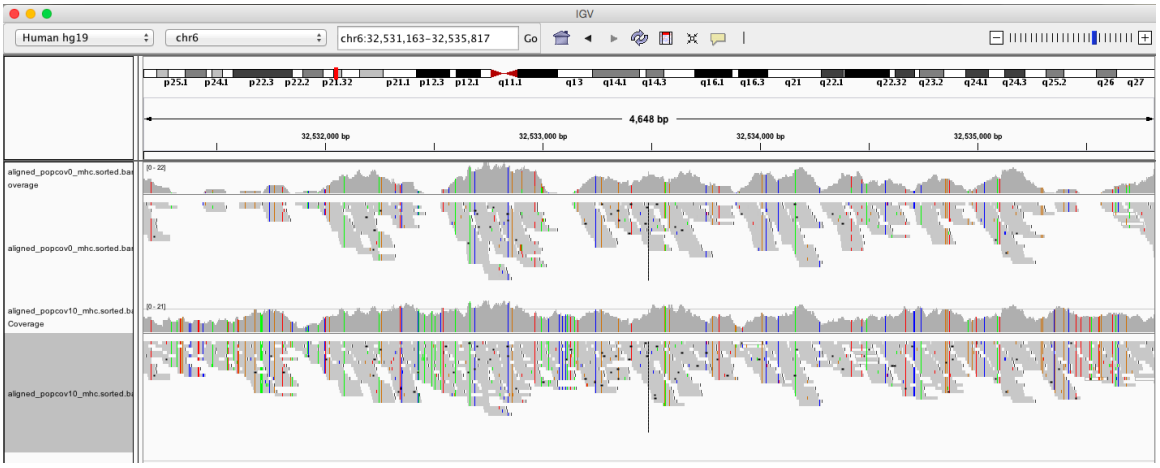


Figure 3.8: Substantial improvements in alignment coverage in the MHC region when variants are included in the graph genome. Simulated reads from chromosome 6 of NA12878 were aligned against the linear reference genome (top) and the graph genome including the top-scoring 10% of SNVs ranked according to the *Pop Cov* model (bottom). Visible in the pileup are are multiple places where linear-genome coverage decreases to 0, but graph-genome coverage remains high, because the graph genome includes some or all of the variants in the dense clusters.

### Ethnicity specificity

We also studied how ethnicity-specific augmented references, advocated in other studies [82, 83, 84], can improve alignment. We used FORGe to select variants from two lists: one with variants drawn from and scored with respect to the overall 1000-Genomes phase-3 callset, and another drawn from and scored for just the CEU individuals. In both cases, variants private to NA12878 and family members were excluded and reads were simulated from NA12878.

Figure 3.9 shows alignment rate and correctness when aligning to CEU-specific and pan-ethnic references. As expected, the CEU-specific reference yielded higher alignment rate and correctness. CEU-specific curves also peaked at lower numbers of SNVs compared to pan-ethnic. However, the differences were only a few hundredths of a percentage point and cover only a small fraction of the remaining distance to the ideal point. Looking at this another way, if we extrapolate the results to a whole-genome DNA sequencing experiment with 40-fold average coverage, around 250,000 alignments would be affected. We return to these small differences in the Discussion section.

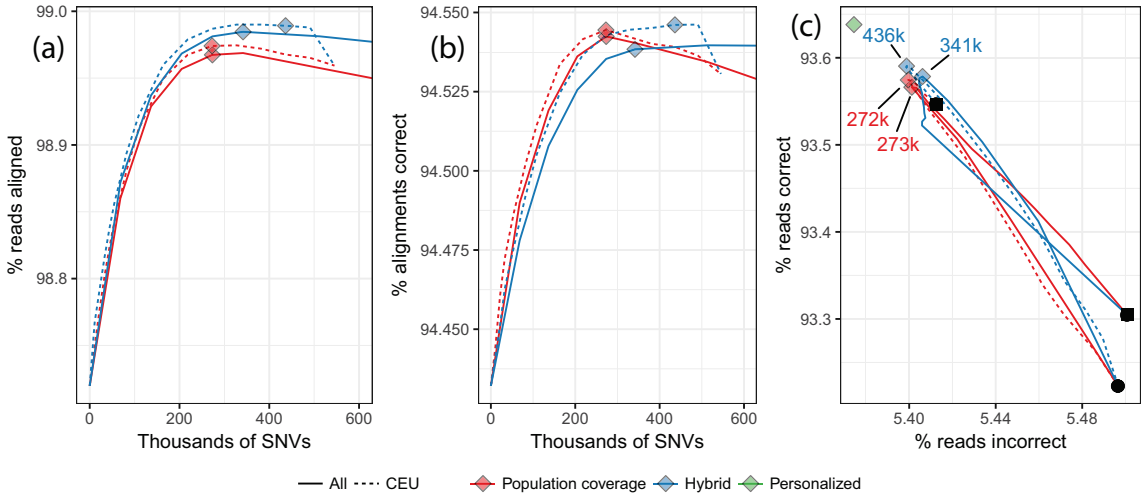


Figure 3.9: Results from chromosome-9 NA12878 simulation when using an ethnicity-specific ("CEU") versus a pan-ethnic ("All") augmented reference. Reads are 100 bp and unpaired.

### 3.3.2 Whole human genome

#### Simulated reads

To show our methods generalize to whole genomes, we repeated experiments like those presented in Figure 3.3 using the full GRCh37 reference. We gathered over 79.2 million SNVs from the Phase-3 callset of the 1000 Genomes Project [6]. We used FORGe's *Pop Cov+* model to score SNVs and compiled subsets consisting of the top-scoring 2%, 4%, 6%, 8%, 10%, 15%, and 20% up to 100% in 10 point increments.We built graph-genome indexes for each using HISAT2. We used the *Pop Cov+* model because the others required excessive time and/or memory; specifically, the *Pop Cov* model (without blowup avoidance) produced a set of variants that HISAT2 was unable to index in a practical time and space budget. Specifically, even with 1 TB of RAM HISAT2 was unable to build a graph-genome index for the top-scoring 10% of SNVs from the full genome, chosen according to the population coverage strategy. After incorporating the blowup avoidance re-ranking strategy into FORGe, HISAT2 was able to construct the index for up to 80% of SNVs with less than 210 GB of RAM. For 90% of variants or higher, index construction still exhausted memory and failed.

In addition, the *Hybrid* and *Hybrid+* models required excessive time for the step that generates the FASTA file for $G*$ due to exponential blowup. The *Hybrid* ranking strategy requires that we iterate over all $s$-mers in $G*$ in order to count each $k$-mer's frequency with Jellyfish. The time required to perform this count limits makes the *Hybrid* much less practical than *Pop Cov*. In future work, it will be important to

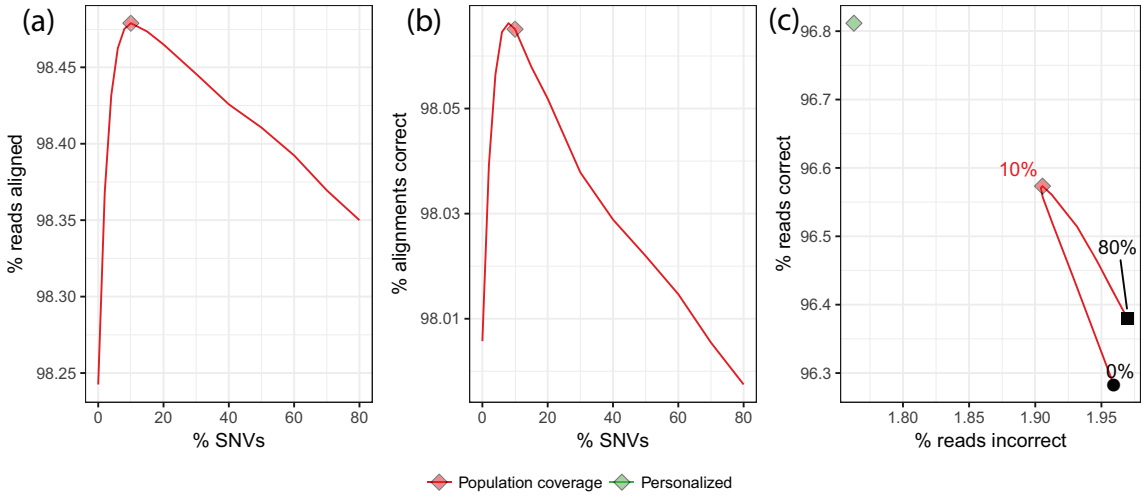


Figure 3.10: Results from aligning NA12878-simulated reads to HISAT2 graph genomes with variants selected using FORGe's *Pop Cov+* model.

investigate ways to make the *Hybrid* strategy more efficient, which could possibly be achieved by switching to a more efficient and/or less precise $k$-mer counting strategy

Figures 3.10a & b plot HISAT2 alignment rate and correctness as a function of the SNV fraction. We aligned 20 million 100 bp unpaired reads from simulated from NA12878. We omitted NA12878 and family members from variant selection. Results using the ideal personalized index are also shown for comparison. Maximal $y - x$, where $y$ is the fraction of reads aligned correctly and $x$ is the fraction aligned incorrectly, occurred at 10% of SNVs (Figure 3.10c). Interestingly, the maximal point does not approach the personalized-genome ideal point as closely here as it did for the chromosome-9 experiment (Figure 3.3). This seems to be due to the added ambiguity that comes when variants in all non-chromosome-9 portions of the genome are added.

**Platinum reads, SNVs**

For a more realistic setting, we conducted further experiments using a set of 1.57 billion real 100 bp unpaired sequencing reads from the Platinum Genomes Project [85] (accession: ERR194147). Like the simulated reads, these also come from NA12878. For this experiment we gathered a set of 80.0 million SNVs from the 1000 Genomes phase-3 callset but omitting variants private to NA12878 and family members. We again used the *Pop Cov+* model to select variants.

We cannot assess correctness since the reads were not simulated. Following a prior study [86], we measured the number of reads that align uniquely — where HISAT2 reported exactly one alignment — versus the number that aligned perfectly, matching the reference exactly with no differences. The goal was to capture the variant-inclusion trade-off; we hypothesized that adding more variants will remove the alignment-score penalty associated with known genetic variants (increasing the number of perfect matches) without increasing reference ambiguity (decreasing the number of unique alignments). As shown in Figure 3.11a, the points that achieved the peak number of unique plus perfect alignments corresponded to 30% of the SNVs. This fraction is higher than most of our simulated results, perhaps due to the fact that unique-plus-perfect is an imperfect proxy for correct-minus-incorrect.

### Platinum reads, SNVs and indels

To highlight the effect of including indels in the reference, we repeated the previous experiment but using both SNVs and indels from the 1000 Genomes phase-3 callset. Specifically, we gathered 83.1 million variants, both SNVs and indels, but omitting variants private to NA12878 and family members. We again used the *Pop Cov+* model to select variants. We again plotted the number of reads that aligned uniquely versus the number that aligned perfectly (Figure 3.11a). The graph genome built from both SNVs and indels achieved peak unique+perfect at 30% of variants, like the graph built from SNVs alone. However, at every percentage it yields more unique and perfect alignments.

### Reference bias

We measured how reference bias varies with the fraction of variants included. We analyzed the alignments of the ERR194147 reads to the whole human genome with both SNVs and indels included in in reference. Figure 3.11b shows a series of boxplots summarizing bias at a set of 2.07 million HET SNVs called in NA12878 by the Platinum Genomes Project [85]. The set of 2.07M HETs was chosen by taking all HETs covered by at least 25 reads in all of our experiments. Each boxplot summarizes the fraction of REF alleles ($REF/(REF+ALT)$) at the HET site for all 2.07M HETs. As expected, bias decreased as more variants were included. The decrease plateaued at 10–20% of variants. Beyond 20%, including more variants did further reduce bias,

but only slightly. From 20% to 70% of variants the mean decreased by only 0.00011. This is consistent with previous results showing that most of the benefit is achieved at a small fraction of variants.

#### HLA typing accuracy

Finally, to measure how the graph genome affects relevant downstream results, we measured how HLA typing recall and accuracy vary with the fraction of variants included in the reference. After observing that the inclusion of about 10% of the variants lead to a qualitative improvement in the alignment pileup (Figure 3.12) we used the Kourami [87] HLA typing tool to make HLA calls at all variant-inclusion fractions...

## 3.4 Discussion

FORGe's modeling of positive and negative effects of including genetic variants in an augmented reference yields accuracy-blowup tradeoffs superior to current approaches. We proposed models for prioritizing variants with distinct rationales and strengths. We found repeatedly that the most advantageous set of variants consisted of a fraction (6–30%) of the variants called in the 1000 Genomes project. In the case of the *Pop Cov* model, this corresponded to an allele frequency threshold of around 3.8–5%. Though the best threshold varied depending on the properties of the read aligner used, we suggest a rule of thumb of filtering out variants with allele frequency

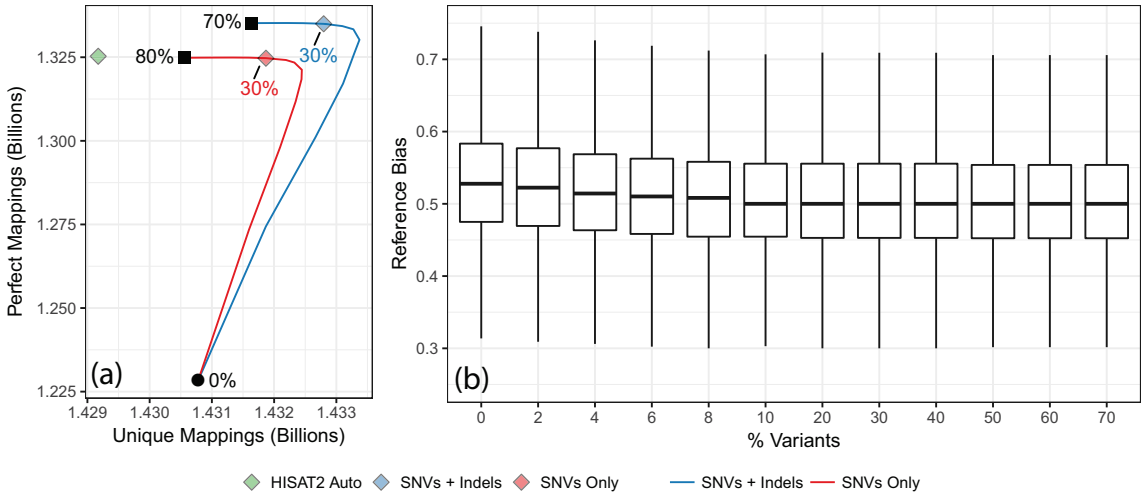


Figure 3.11: (a) Perfect/unique alignment results when aligning real reads. The blue curve is parametric, as a function of the fraction of variants included from 0% (bottom left) to 80% (top). The green diamond marks the number perfect and unique mappings for HISAT2's custom variant pruning script applied to the set of SNVs and Indels. Graph genomes were built for SNVs alone (red) and for SNVs and Indels (blue), both ranked with the *Pop Cov+* strategy. Blue and red diamonds mark the fractions that achieved the highest sum of unique and perfect alignments. (b) Allelic bias for the 2.07 M heterozygous SNVs that met a minimum coverage threshold of 25 in all experiments. Whiskers show the 5th-to-95th percentile range.

under 5%, as was done in at least one prior study [27]. We also showed that FORGe's modeling can substantially reduce reference bias, also at a relatively low fractions of variants included.

FORGe and HISAT2 combine to make a practical graph aligner that works with human data with large variant databases like the 1000 Genomes Phase 3 call set. Using `hisat2-build` to index a GRCh37-based graph genome with the top 8% of variants from Phase-3 set required 4 hours and 165 GB of memory. Aligning 20 million reads to this graph required 19 minutes and 6.5 GB of memory, about 50% more time

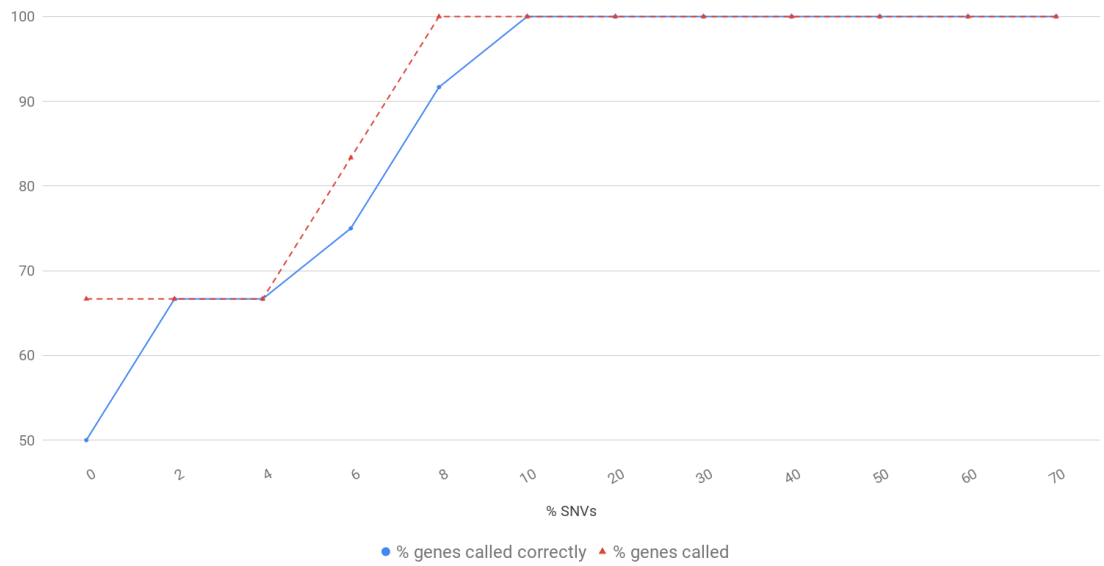


Figure 3.12: HLA typing results called with Kourami from the same set of real reads in figure 3.11. The percentage of genes called (red) and correctly called (blue) are both shown.

and 50% more memory than HISAT2 requires to align to the linear GRCh37 genome. (To prioritize the variants prior to indexing, FORGe required about 110 minutes on a single processor.) This is competitive with the performance of aligners like Bowtie 2 and BWA-MEM when aligning to the linear reference, suggesting graph-based tools are ready for broader use.

Though we estimate that the overall improvement in alignment accuracy for a 40x whole-genome DNA sequencing experiment would lead to 4.8M more correctly aligned reads and 1.2M fewer incorrectly aligned reads, the magnitude of the improvement imparted by modeling variants depends on the genomic region. For some

regions and variant classes (rare, isolated SNVs), the benefit is small. To improve alignment to these regions might require an iterative approach that aligns to a graph containing known variants, calls donor-specific variants, then realigns to a graph that includes both. Strategies like this are implemented in the GATK HaplotypeCaller [67], GraphTyper [68] and other tools [88]. Better variant models might also benefit these hard cases. Even so, the effects we measured translate into substantial net increases in the number of correctly aligned reads, and the results are pronounced in regions such as MHC as shown in Figures 3.8 and by the HLA typing experiment (Figure 3.12).

An ethnicity-specific reference conferred a slight accuracy improvement compared to a pan-ethnic reference with a similar number of variants. This is notable in light of proposals to use ethnicity-specific references [82, 83]. It suggests that the advantages of an inclusive reference, applicable regardless of the donor individual's ethnicity, might outweigh the slight accuracy gain that comes with ethnicity-specificity. Also, ethnicity-specific references could be counterproductive or misleading in cases where donor ethnicity is reported incorrectly or where the donor is admixed [89].

The accuracy achieved at relatively small fractions of the 1000 Genomes variants has implications for the design of graph aligners. A central challenge for these tools is to operate efficiently even when variants are densely clustered, causing local explosion in the number of allelic combinations. But our observations that peak accuracy occurs at a relatively small fraction of variants, and that memory footprint increases by a

factor of 2 or less at peak accuracy, suggests that this is not a major barrier to practical graph-genome alignment as long as variants are chosen carefully.

It should also be possible to adapt FORGe to study how including structural variants can improve alignment. A common observation of studies that have assembled human genomes from long reads is that the assemblies contain many megabases of sequence not present in the standard human reference [83, 84, 90, 91]. The models we propose are equally applicable to structural variants, assuming the variants are called in enough individuals to estimate allele frequencies accurately.

# Chapter 4

# CNV Analysis with Ginkgo

Since the initial introduction of single cell DNA sequencing that required a manual preparation for each cell sequenced, the number of cells that can be analyzed in a single experiment is rapidly growing. Recently, 10X Genomics (Pleasanton, CA) announced the release of a new single cell CNV kit that, among other advances, significantly increases the number of individual cells that can be barcoded and sequenced in a single experiment. While Ginkgo was designed for dozens to hundreds of cells, we now anticipate datasets containing thousands to tens of thousands of cells that must be processed in tandem will become routine.

In light of the increasing scale of single-cell experiments using the new 10X Genomics kit and related high-throughput approaches, we have added several features to Ginkgo for more effective and convenient usage, released in Ginkgo 2 (Figure 4.1). Most significantly, we present two new methods that improve accuracy even

with relatively low coverage. These methods improve the localization of copy-number transitions, reducing the number of incorrect copy-number assignments for genes that lie near such transitions. First, Ginkgo 2 can aggregate data across cells with similar copy number profiles, increasing read depth through "borrowing" and enabling a smaller bin size. Second, Ginkgo 2 can selectively introduce smaller bins near copy number changes to improve localization without significantly increasing noise across the full genome. We also introduce a number of improved user interface tools for more effective transfer and analysis of large datasets as well as a novel binning technique to avoid unmappable regions more robustly. We apply these improvements to several real and simulated single cell datasets to highlight the improvements to accuracy made possible by these methods.

## 4.1 Methods

### 4.1.1 Improved Binning Technique

As the typical depth of coverage in a single cell DNA analysis is very limited, a core component to any copy number analysis method is a strategy for aggregating (binning) read counts across larger segments of the genome. The original Ginkgo provides a set of variable-length bins defined such that each bin contains the same number of *uniquely mappable* bases. A base is considered *uniquely mappable* for a given read length $r$ if a read of length $r$ beginning from that base aligns uniquely and correctly

to that position in the reference genome. Most of the human genome as well as most other large genomes consist of uniquely mappable bases, interspersed with sporadic unmappable bases due to repetitive or undefined regions (Ns and other sequencing gaps). However, there are also occasional large unmappable regions, most notably the centromeres of each chromosome, that create extended regions where no reads can be reliably mapped. For example, Figure 4.2 (top) shows the composition of bins created by Ginkgo for a desired bin size of 1Mb. Aside from the clearly-identifiable centromere, there are multiple other poorly mapped bins containing almost 1 million unmappable bases throughout the chromosome. Such bins are not only more likely to skew segmentation results with abnormal read counts, but larger bins make copy-number changes within these bins harder to localize accurately. Ginkgo 2 introduces a new method for generating variable-length bins that avoids creating bins that contain long regions of poor mappability (Figure 4.3). We describe both the original and the new bin-generation methods below.

**Original Bin-Generation Method**

Ginkgo's original bin-generation algorithm takes as parameters a genome, a list of genes, and a read length $r$, bin width $b$, and aligner (BWA [9] or Bowtie [8]). The script then performs the following steps for each chromosome:

1. Assign each base in the genome a *mappability* flag indicating whether a read from this position is uniquely mappable. To do this, sample an unpaired read

of length $r$ starting at that position and align it to the full genome. If the read maps uniquely to the correct position, set the mappability flag to 1. Otherwise set the flag to 0.

2. Beginning from the leftmost mappable base, iterate through all bases in the genome while incrementing a counter for each mappable base. When the counter reaches $b$, create a new bin beginning at the end of the previous bin and extending to the current base, and reset the counter to 0. When the number of mappable bases in an arm of the genome is not evenly divisible by $b$, the bin furthest from the centromere should be extended to contain all remaining mappable bases.

This method excludes the telomeres and guarantees that all bins contain the same number of mappable bases, except possibly the last bin on the extreme end of each chromosome. Thus, where the copy number remains constant (and assuming no biases affecting coverage level) we would expect the same number of reads to map to each bin. Unfortunately, bins that span large unmappable regions may be much longer than the desired bin width. In particular, one of the bins bordering the centromere will contain the centromere itself. For example, the bins for chromosome 1, computed for $b = 1$Mb, include a bin surrounding the centromere of length 26Mb, for which only 4% of the bases are mappable (Figure 4.2, top). Copy number changes that occur near unmappable regions may not be well localized, and the very large genomic span can mask the presence of CNVs. Ideally, long stretches of unmappable bases should

be segregated into separate bins to avoid confusion with mappable regions.

### New Bin-Generation Method

Ginkgo 2 implements a new method that avoids creating bins spanning large unmappable regions. In addition to the parameters required for the original method above, the new method includes parameters $i$ and $p$ for the minimum *island size* and *unmappable island threshold*, described in more detail below.

1. Assign each base in the genome a *mappability* flag, using the same method as step 2 of the original binning method described above.

2. Identify large regions of poor mappability, which we call *unmappable islands*. To find these islands, split the genome into non-overlapping windows of length $i$ and find all windows for which at least a proportion $p$ of bases are unmappable. If any of these windows are directly adjacent, merge them.

3. Beginning from the left, iterate through all bases in the genome while incrementing a counter for each mappable base. When the counter reaches $b$, or the edge of an *unmappable island* is reached, create a new bin beginning at the end of the previous bin and extending to the current base, and reset the counter to 0. If the newly-created bin is less than $\frac{b}{2}$ bases in length and is preceded by a bin (not an island), merge it with the previous bin. Each bin is stored with the number of uniquely-mappable bases it contains.

The new binning method is illustrated in Figure 4.3. Bins created with this method will by construction avoid spanning large unmappable regions, though in some cases a bin may contain a large proportion of unmappable bases that happen to be dispersed evenly throughout it. We can guarantee that no bin contains a proportion of $\geq p$ unmappable bases, since in such a case the bin would overlap at least one unmappable island. In practice our approach is highly effective for separating mappable and unmapped regions. For example, Figure 4.2 demonstrates the improvement in bins produced by the new method. Notably the proportion of unmappable bases is higher for bad bins and lower for good bins. We also find that in practice the maximum proportion of bad bases in all good bins is often substantially lower than the maximum guaranteed by our parameters.

Unlike the original binning method, the new method creates bins with potentially variable numbers of mappable bases, especially those bins adjacent to an unmappable island. Consequently Ginkgo 2 records each bin's unmappable base count $c$ and divides each its read count by $c$ to obtain a normalized count before segmentation. Ginkgo 2 also stores each unmappable island as a bin, but marks them with an *unmappable* flag. Ginkgo 2 removes these unmappable bins before segmenting the bins by copy number.

### 4.1.2 Aggregating Read Counts from Similar Cells

Particularly in large experiments, we often find that many cells appear to be closely related with nearly-identical copy-number profiles. In a cancer context these would include multiple cells derived from the same clone, or normal cells that all share the same (germline) CNVs. In such cases, Ginkgo 2 can improve segmentation by borrowing strength from "replicate" cells. Merging read counts from multiple cells allows Ginkgo 2 to reduce the bin size, leading to more accurate localization of copy number changes: aggregating 10 cells can increase the power to detect CNVs that are up to 10 times smaller.

Ginkgo's original release includes a cell clustering visualization (Figure 4.1d) based on the similarity of the copy number profile of the different cells in the analysis. Ginkgo offers a number of clustering methods as options for the user, including single linkage, complete linkage, ward linkage or neighbor-joining, as well as several options for the distance metric including Euclidean, Manhattan or Minkowski distances. After computing copy number profiles for all cells, Ginkgo uses the specified clustering method to construct a similarity tree of all cells. In Ginkgo 2, we leverage this clustering to find replicate cells for aggregation.

A similarity score between pairs of cells in the clustering can be inferred from the height of the lowest branch point that contains both cells. Accordingly, a user may choose a similarity threshold below which most cells can be treated as replicates. Ginkgo 2 generates clusters by cutting the tree at the height of the given threshold

(Figure 4.1e). For each cluster, a new synthetic *aggregate cell* is created that contains all the reads from each cell in that cluster. Ginkgo then recomputes the copy number profile for each of these *aggregate cells*. The user may also optionally specify a different bin size to use for the second segmentation; if many cells are combined into a single cluster, a much smaller bin size can be used for greater accuracy.

The Ginkgo 2 website includes an interactive tree that visualizes the effects of cutting the tree at any given height. Choosing an accurate threshold for clustering is an essential part of this process. Using too high of a threshold will lead to merging cells with substantially different copy number profiles, effectively averaging their copy numbers and potentially obscuring important data. Too small a threshold will fail to take full advantage of "replicate" cells, and thereby limit the CNVs that can be detected. Ginkgo 2 auto-fills the threshold input with a suggested threshold (defined below), although the user may also specify their own threshold after inspecting the copy number profiles.

Ginkgo 2 computes a suggested threshold at which to cut the tree based on the principle that the copy-number profiles for replicate cells should differ by only a few bins at the boundaries of where the copy number changes since these bins are most sensitive to noise, but should not differ by long contiguous stretches. Specifically, the suggested threshold is the largest height at which no pair of cells assigned to the same cluster differ in their copy number profile by at least 30 contiguous bins. The default threshold does not claim to be the optimal threshold for every sample, but is simply

a starting point to guide the user in choosing an appropriate threshold.

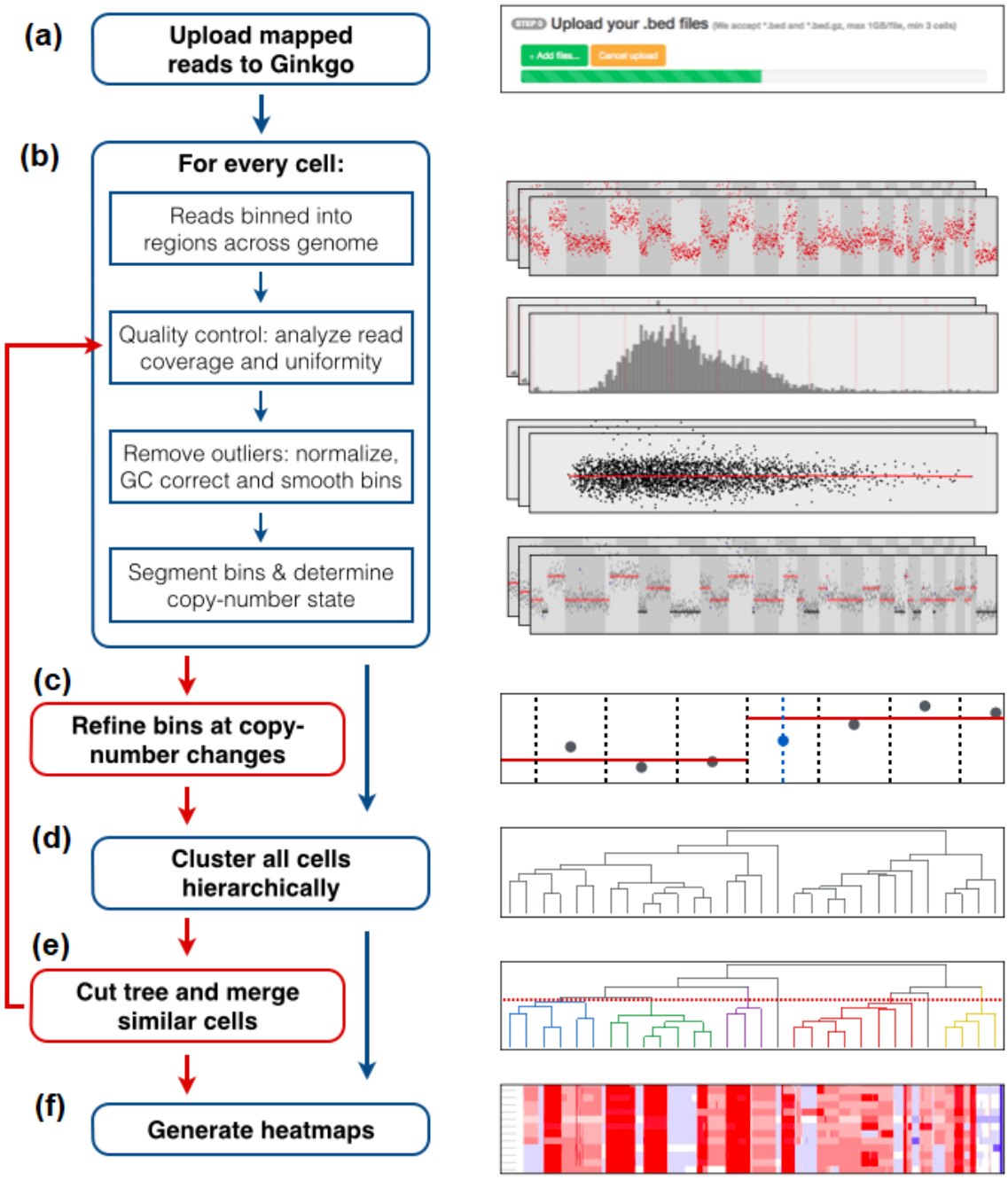


Figure 4.1: The Ginkgo flowchart for performing single-cell copy-number analysis. New methods introduced in Ginkgo 2 are highlighted in red.

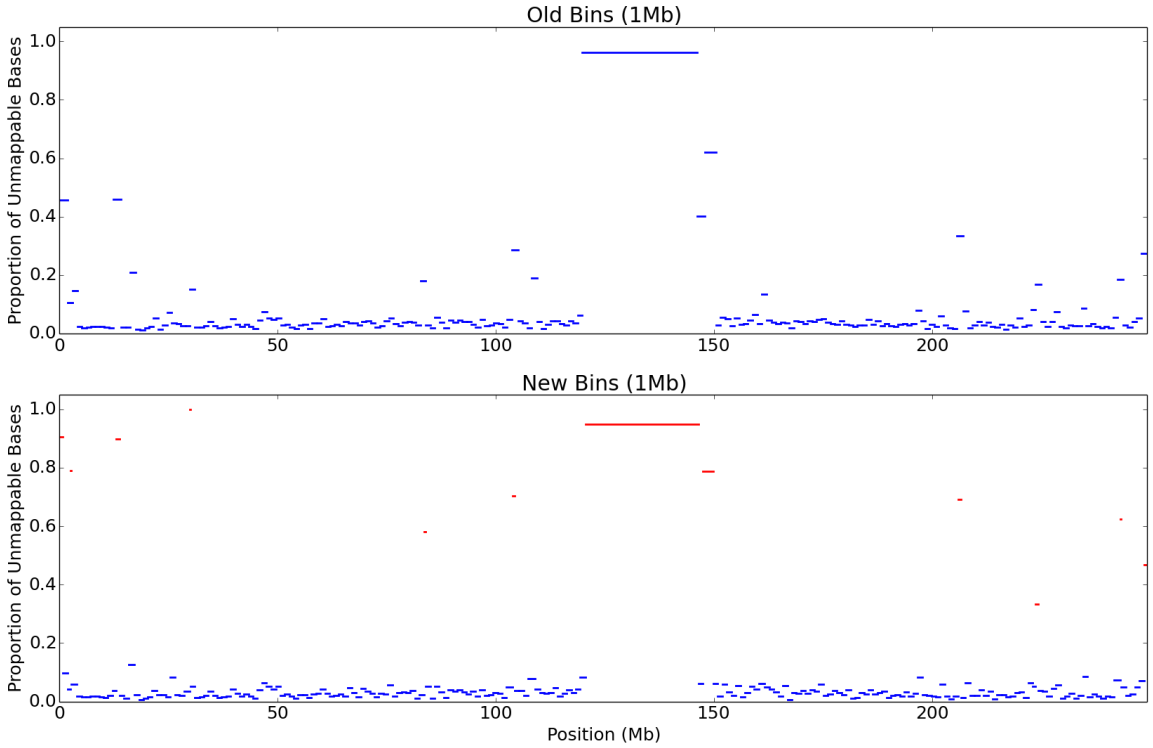


Figure 4.2: Comparison of original Ginkgo 1 bins (top) and new bins (bottom) in human chromosome 1. Red segments indicate unmappable islands and blue segments indicate "good" bins with a high proportion of mappable bases. New bins were generated with a window size of 100kb (0.1X desired bin size) and bad island threshold of 0.5635, thus guaranteeing that no good bin will contain more than half unmappable bases. In GRCh37, we find that the peak proportion of bad bases is $< 0.1253$ for all good bins.

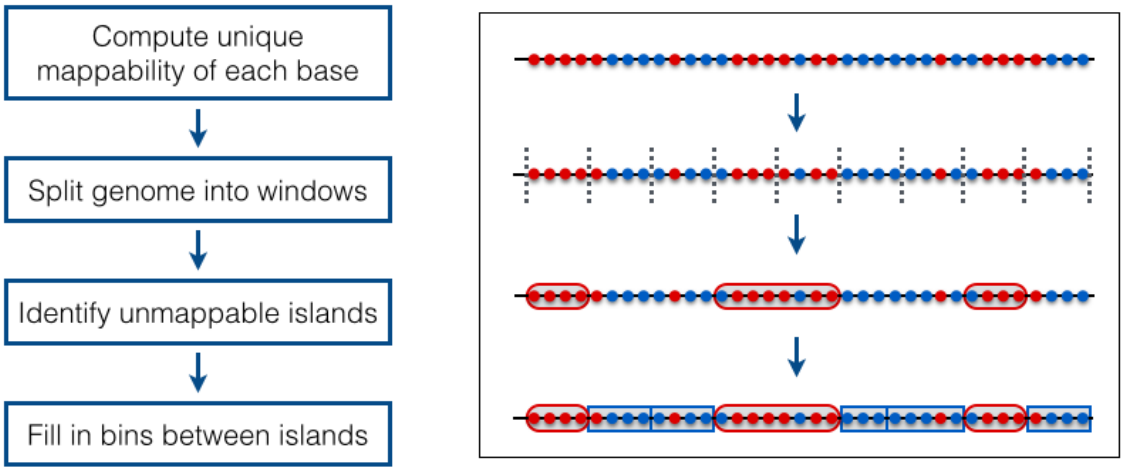


Figure 4.3: The new binning method included in Ginkgo 2. First, each base in the genome is flagged as uniquely mappable or not. Red dots indicate unmappable bases, blue dots indicate mappable. Next, the genome is divided into non-overlapping windows. Windows with a high proportion of unmappable bases are classified as non-mappable islands and adjacent islands are merged. Finally, bins are constructed between islands such that every bin contains between $\frac{b}{2}$ and $\frac{3b}{2}$ mappable bases, where $b$ is the desired bin size.

### 4.1.3 Bin Refinement at Copy Number Changes

Ginkgo, as with other CNV analysis algorithms, allows copy number changes only at bin boundaries. Consequently, even when CNVs are correctly identified, the results can be misinterpreted when a transition is truly in the middle of a bin. For a typical bin size of $\sim 1$ *Mbp* bins, the true transition may therefore be as much as 500kb away from a boundary, a difference that can easily affect the copy number status of one or more genes. Furthermore, when the true copy number change occurs inside a bin, the bin's average count tends to lie between those of the flanking bins (Figure 4.4).

In Ginkgo 2, read counts are stored not only for the specified bin width, but also for the smallest available bins, currently 5kb, that we call *fine bins*. The CNV analysis algorithm uses the counts in the fine bins to determine the position of the larger CNVs. Let $r$ be the average number of fine bins contained in each large bin. If Ginkgo detects a copy number change between bins $B_1$ and $B_2$, we assume that the true change occurs somewhere between the start of $B_1$ and the end of $B_2$. Let $C_1$ and $C_2$ be the average read counts per bin of the segments ending with $B_1$ and beginning with $B_2$, respectively. Let $b_1, \ldots, b_n$ be the fine bins fully contained in $B_1$ and $B_2$, with read counts of $c_1, \ldots, c_n$ respectively. For each fine bin $b_k$, we compute the weighted counts of the left and right sets of fine bins as

$$count_{left} = \frac{r}{k} \sum_{i=1}^{k} c_i$$

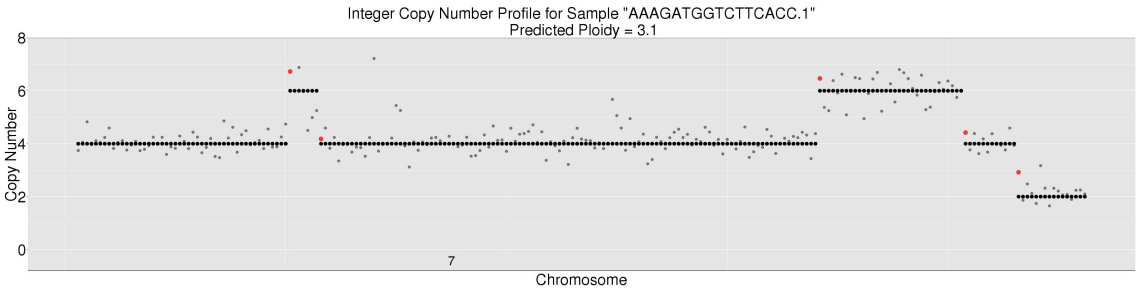


Figure 4.4: Sample initial copy-number profile for 500kb bins in chromosome 7. Normalized read counts for each bin are marked in gray, assigned copy-number states in black, and identified outlier bin counts in red.

$$count_{right} = \frac{r}{n-k} \sum_{i=k+1}^{n} c_i$$

And the *boundary score* is the sum of squares of the distances between the left and right fragments and the average read counts of their adjoining segments:

$$\text{boundary score} = (count_{left} - C_1)^2 + (count_{right} - C_2)^2$$

Ginkgo 2 chooses the fine bin break point that minimizes this boundary score, leading to improved accuracy at the ends of CNV events. Note that this method is still susceptible to variation in the reads counts of fine bins, especially at low coverage, and may overfit bin boundaries to noise in the read counts. However, such overfitting is limited to the neighborhood of copy-number changes, and in general this method improves the precision of segmentation.

### 4.1.4 User Interface Improvements

The distribution of reads across the genome is non-uniform, due to factors such as random primer amplification, sequencing bias, and repetitive or unmappable regions. The low coverage of single-cell sequencing is more severely affected by noise, which adds increased dispersion over bulk sequencing datasets with deep coverage. Ginkgo 2 offers new *minimum bin width* and *maximum ploidy* parameters to help deal with noisy data. Both parameters can be changed in the *Advanced Parameters* section of the interactive website.

The *minimum bin width* parameter specifies the minimum number of bins allowed in any copy number segment as computed by the circular binary segmentation algorithm in DNACopy [92]. A small value will often lead Ginkgo to discover more unique copy number states, but is more susceptible to over-fitting to the noise in the data.

The *maximum ploidy* parameter sets an upper bound on the ploidy chosen for the final copy numbers. After segmenting the normalized read counts, Ginkgo chooses the scaling factor that minimizes the sum of square differences between scaled counts and integer values. Maximum ploidy sets an upper bound on the scaling factors tested. A lower value reduces the chance of choosing a large outlier at high ploidy, which would lead to inflated copy number values.

## Scalability Improvements

The original Gingko required users to upload a single bed file for each cell, so that samples with 1000s of cells would require 1000s of separate uploads. Ginkgo 2 allows the user to upload data from many cells in a single bed file, provided each read is tagged with its cell of origin in the rightmost column. The user should indicate that a file contains multiple cells by uploading a file with the same name and an added '.cells' extension, containing a list of all cells from the bed file that should be processed. Ginkgo 2 also adds the option to download a tar file containing all CNV profile images to avoid downloading each image independently.

## Gene Copy Number Heatmap

Ginkgo 2 adds a fourth heatmap to the three previously displayed on the main results page. This heatmap shows the copy number assigned to all genes in the COSMIC list of cancer-associated genes curated by the Cancer Gene Census [93]. For this figure we use the Tier 1 gene list from `https://cancer.sanger.ac.uk/census`. Figure 4.5 shows one such heatmap for a set of cells from the the SK-BR-3 cancer cell line.

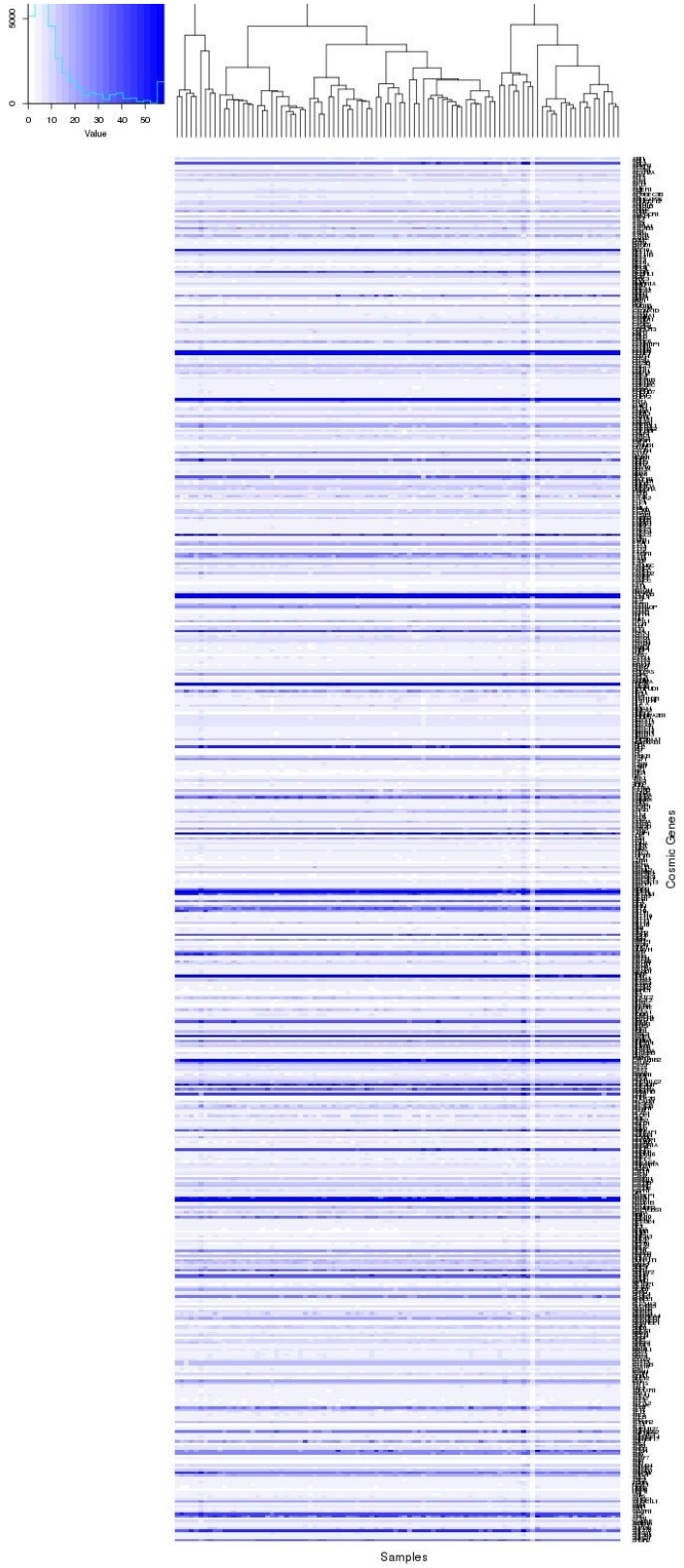


Figure 4.5: Copy number assigned to all genes in the COSMIC gene list associated with cancer. For genes that span multiple distinct copy number states, reported copy number is the average copy number across all bases in the gene.

## 4.2 Results

### 4.2.1 Datasets for Testing

**Simulated Data:** We simulated an aneuploid human genome based on GRCh37 with 120 simulated CNVs with an average CNV occurrence rate every 25 million bases. CNV lengths were drawn from a log-normal distribution with mean of 10 million bases and ranging from 5kb to 200Mb in length. While these parameters may not truly represent the true distribution of CNVs in every experimental dataset, the high occurrence rate and wide range of sizes allow us to thoroughly test Ginkgo's flexibility and accuracy. CNVs were defined by adding or removing contigs in the genome fasta (e.g. a normal diploid genome fasta would contain two copies of each contig.)

We then used Mason [79] to simulate 2 million reads from this simulated genome (approximately 0.6x coverage). We aligned these reads to GRCh37 with Bowtie [8], and selected the roughly 1.65 million reads which aligned uniquely as our read set using samtools [10]. We measured the accuracy of each CNV segmentation by the proportion of bases in the genome which were assigned the correct copy-number.

Figure 4.6 shows the per-base accuracy of CNV calls for various bin sizes and down-sampled subsets of the simulated read dataset. As a general rule of thumb, Ginkgo achieves optimal accuracy when bins are as small as possible while maintaining an average of at least 20 reads per bin. At extremely low coverage levels or large bin

sizes, accuracy begins to decline as either noise begins to overwhelm the read depth analysis or the large bins dilute the signal from small CNVs .

**SK-BR-3 Sequencing Data:** We also analyzed genuine single-cell data from the SK-BR-3 breast cancer cell line. This dataset contained single-cell sequencing data from 96 unique cells, as well as a BAM file containing approximately 800 million aligned reads from a bulk DNA sequencing experiment. We discarded 2 of the cells which contained less than 10 thousand reads each. All SK-BR-3 experiments below used the remaining 94 cells, which contained an average of 1.2 million reads each.

Since this data is not simulated, there exists no ground truth to test the accuracy of Ginkgo's CNV profiles. Instead, as a proxy for correctness we compare CNVs assigned to genes to their average read coverage from the bulk sequencing dataset. Specifically, for each gene in the Ensembl GRCh37 annotation, we compute:

1. The average per-base read coverage of bulk sequencing alignments across all bases in that gene.

2. The average copy-number assigned to all bases in the gene. Usually the entire gene is contained in a single copy-number state, in which case we assign the gene that copy number. In the infrequent case where the gene spans multiple copy-number states, we compute a weighted average of the different copy-numbers, weighted by the number of bases that they span.

After determining these two values for all genes, we compute the Pearson corre-

lation between the vector of coverages and vector of copy numbers. In general, we expect the batch sequencing coverage at any point in the genome to directly correlate with the true copy number at that point. Figure 4.7 shows the average correlation of all 94 cells for various bin sizes and down-sampled read sets. The correlation is lower than the accuracy for the simulated data, as expected since the whole genome amplification used for generating the single cell data will have biases not captured in the simulation. Nevertheless, the curves for each down-sampled dataset resemble the corresponding curves from the simulated data, substantiating our use of this measure as a proxy for correctness.

**MKN-45 Sequencing Data:** We also evaluated a single-cell sequencing dataset from the MKN-45 gastric cancer cell line released by 10X Genomics produced using their new CNV kit. This dataset contains 5,203 cells containing an average of 3.45 million reads per cell. As with SK-BR-3, we approximated accuracy by the Pearson correlation with bulk sequencing coverage for all genes in the Ensembl GRCh37 annotation. We used an existing bulk sequencing dataset of 282 million reads sequenced from MKN-45 (accession number SRR801243 [94]).

Figures 4.6 through 4.8 illustrate the tradeoff between bin size and read coverage mentioned throughout this analysis. In general, accuracy is highest when small bins can be applied to samples with deep coverage, and decreases for larger bin sizes and lower coverage. As a general guideline, a coverage of about 20–30 reads per bin appears to yield the best accuracy for the simulated and SK-BR-3 datasets, a

value consistent with earlier Ginkgo recommendations [39]. The MKN-45 dataset peaks at a higher coverage, around 100 reads per bin due to higher noise in the 10X Genomics scDNA data. Correlation with bulk sequencing is lower for MKN-45 than for SK-BR-3, most likely due to the lower coverage of the bulk sequencing data and higher frequency of extreme copy number states in SK-BR-3. As a second measure of accuracy for this dataset, we computed the CNV for the bulk sequencing data with Ginkgo for 10kb bins and computed the average Euclidean distance between each sample CNV and the bulk CNV (Table 4.1) and see near universal improvements in accuracy using the bin refinement technique.

| Bin Size | Original | Bin Refinement (5kb) | |
|---|---|---|---|
| 10kb | 194,128 | 193,462 | 666 (0.3%) |
| 50kb | 162,399 | 162,570 | -171 (-0.1%) |
| 100kb | 158,970 | 158,950 | 20 (0.01%) |
| 250kb | 161,931 | 161,906 | 25 (0.02%) |
| 500kb | 168,131 | 168,107 | 24 (0.01%) |
| 1Mb | 169,966 | 169,746 | 220 (0.1%) |

Table 4.1: Average Euclidean distance between sample CNVs and bulk CNV computed for 10kb bins, for the original Ginkgo segmentation and with the addition of the bin refinement step.

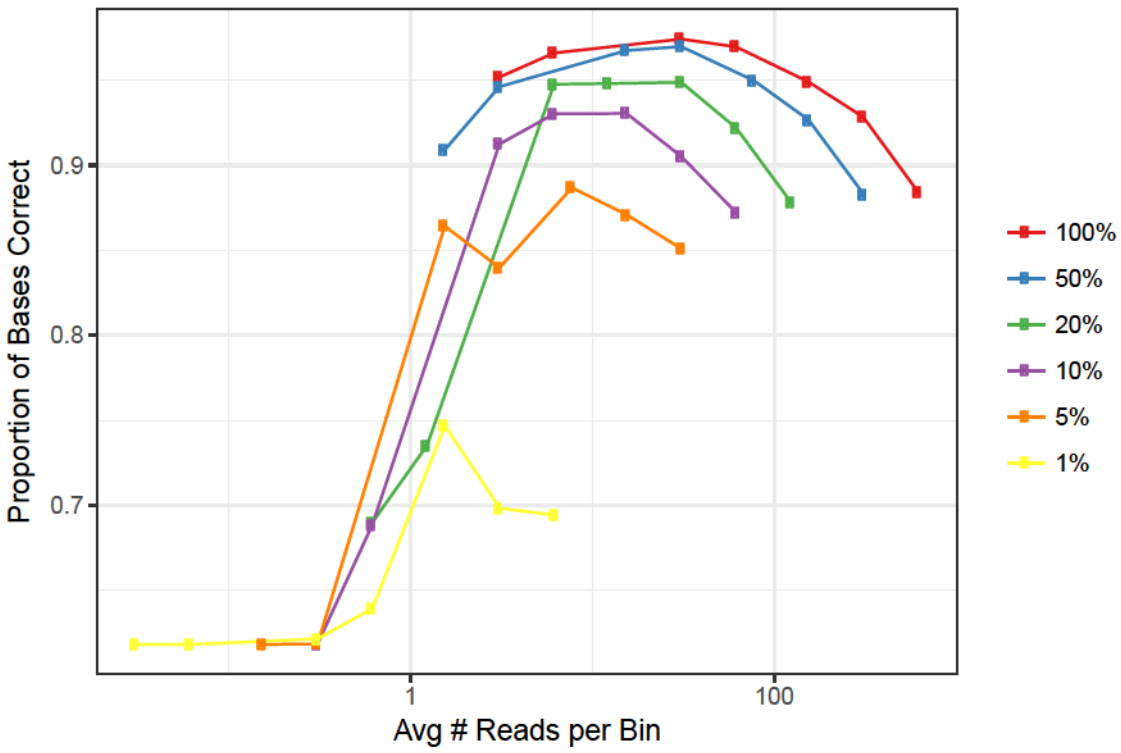


Figure 4.6: Accuracy of CNV segmentation for simulated data across different downsampling rates and bin sizes. Each curve corresponds to a different downsampled dataset. Curves are plotted such that the x-axis marks the average number of reads in each bin. On each curve the bin sizes are, from left to right: 5kb, 10kb, 50kb, 100kb, 250kb, 500kb, 1Mb.

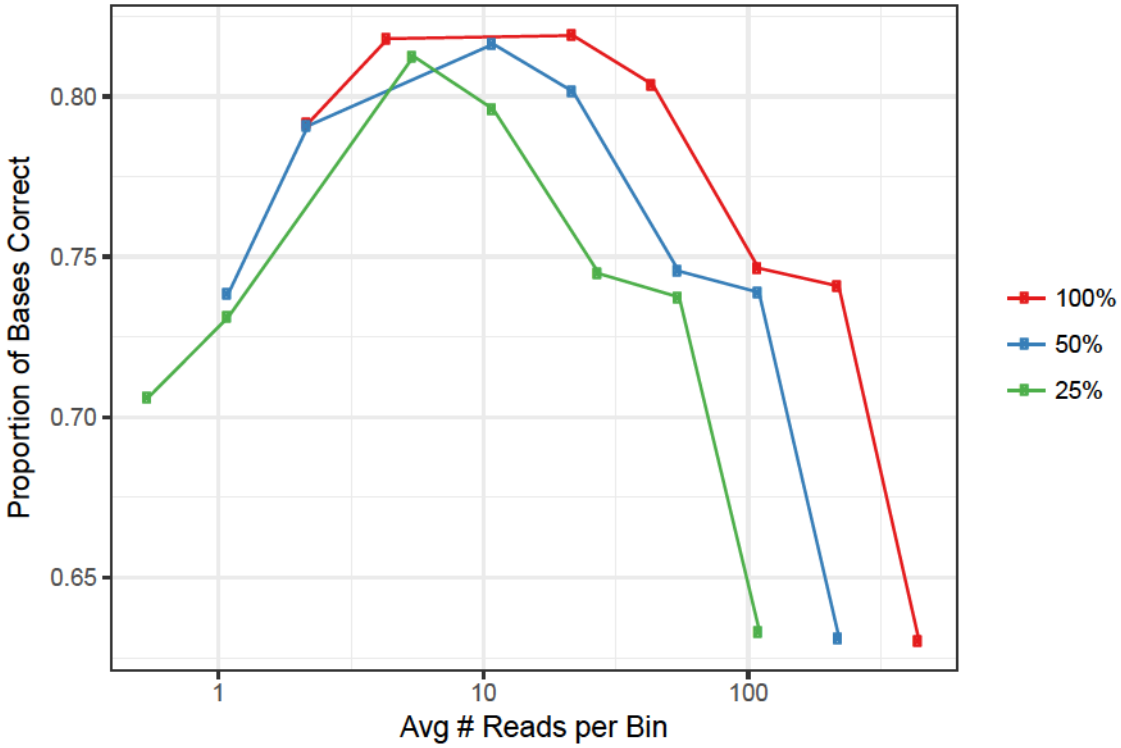


Figure 4.7: Correlation between gene coverage and assigned CNV for SK-BR-3 data across different downsampling rates and bin sizes. Each curve corresponds to a different downsampled dataset. Curves are plotted such that the x-axis marks the average number of reads in each bin. Thus for each curve, the largest bin size (1Mb) is the rightmost point and the smallest bin size (5kb) is the leftmost. Low Pearson correlation is expected due to true differences between the single-cell and bulk gene expression profiles.

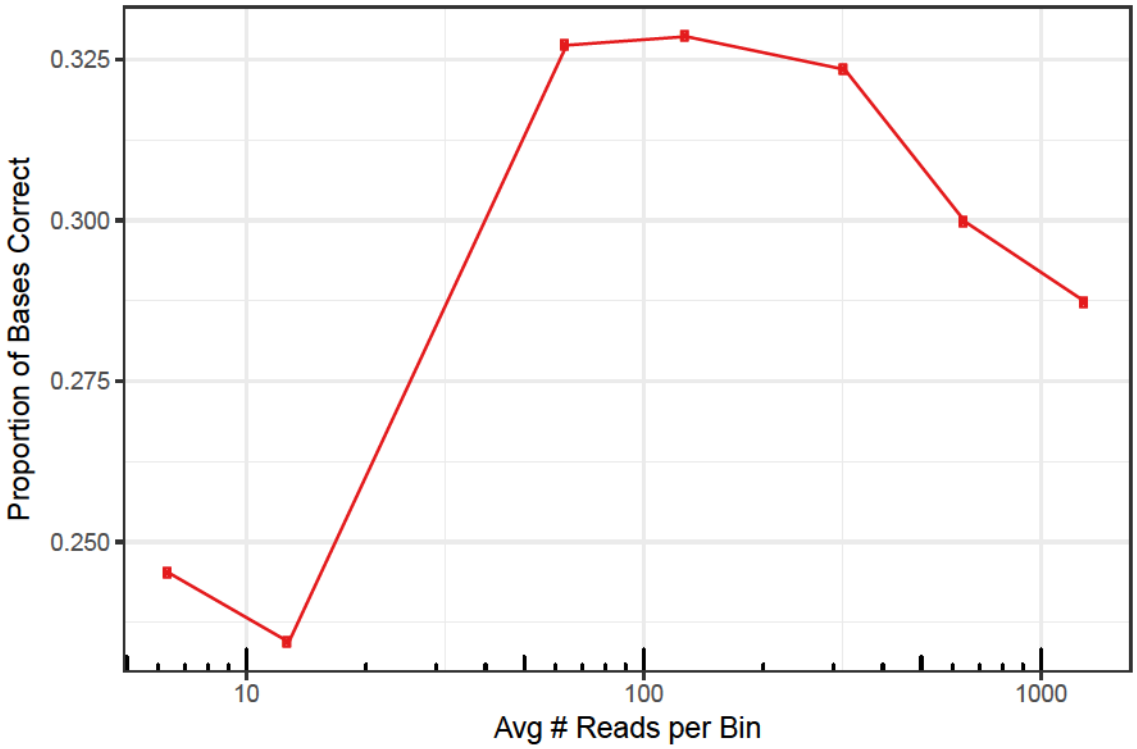


Figure 4.8: Correlation between gene coverage and assigned CNV for MKN-45 data. The largest bin size (1Mb) is the rightmost point and the smallest bin size (5kb) is the leftmost.

### 4.2.2 New Binning Method

We next tested the accuracy of Ginkgo 2's new binning method. By construction, the new method should more accurately exclude regions of high unmappability in bad bins. Figure 4.2 visualizes the change in bins after adoption of the new method. Overall, unmappable bins become smaller and have a higher proportion of unmappable bases, while good bins have a lower proportion of unmappable bases. This indicates that we are more accurately differentiating mappable and unmappable bins.

There are many parameters to consider when generating bins with the new method. *Bin size* refers to the desired number of mappable bases to be included in each bin, just as in the original binning method. *Minimum island size*, $i$, is the size of the initial window used to search for bad islands. *Bad island threshold*, $p$, is the cutoff for the proportion of mappable bases in a bad island. All windows with less than this proportion of mappable bases are classified as bad islands. Tables 4.2 & 4.3 show the CNV accuracy for varying minimum island size and bad island threshold. We found the optimal parameter values to vary between datasets, read coverage, and bin size used. However, accuracy was substantially higher for new bins than for the old ones for all parameter values tested.

Figures 4.9 through 4.11 show the increase in accuracy across multiple coverage levels and bin sizes for the optimal parameter sets in Tables 4.2 & 4.3. Bins constructed with the first set of parameters (left) perform slightly better for medium to large bin sizes, while the second parameter set performs better for smaller bin

sizes. Promisingly, the dataset downsampled by 50% achieves almost the same level of accuracy as the full dataset at its peak, suggesting that improved bins increase Ginkgo's effectiveness on sparse datasets.

While there is some variability in accuracy depending on the parameters, all of the parameters we tested yielded a large improvement over the original Ginkgo bins. While it is impossible to choose a single parameter set that optimizes accuracy for all experiments, since the optimal parameter set varies based on the dataset, coverage, and bin size, by default we have selected parameters that guarantees that no bin contains more than 52.4% unmappable bases. Specifically, for bins provided on the active Ginkgo 2 website we use a minimum island size of $\frac{1}{5}$ the desired bin width and bad island threshold of 0.6.

| | | Bad Island Threshold | | |
|---|---|---|---|---|
| | | 0.5 | 0.6 | 0.7 |
| Min. Island Size (Prop. of bin size) | 0.05 | 0.9756 | 0.9754 | 0.9752 |
| | 0.1 | 0.9754 | 0.9749 | 0.9750 |
| | 0.2 | 0.9741 | 0.9695 | 0.9731 |
| | 0.3 | **0.9846** | 0.9841 | 0.9742 |

Table 4.2: CNV accuracy for the full simulated dataset using bins generated with the new method for a bin size of 100kb and varying *Minimum Island Size* and *Bad Island Threshold*. The peak accuracy is shown in bold. The CNV accuracy for the original 100kb bins is 0.9696.

| | | Bad Island Threshold | | |
|---|---|---|---|---|
| | | 0.5 | 0.6 | 0.7 |
| Min. Island Size (Prop. of bin size) | 0.05 | 0.7489 | 0.7452 | 0.7443 |
| | 0.1 | 0.7455 | 0.7455 | 0.7447 |
| | 0.2 | 0.7459 | 0.7459 | **0.7514** |
| | 0.3 | 0.7501 | 0.7507 | 0.7494 |

Table 4.3: Average CNV correlation with gene coverage for all cells in the SK-BR-3 dataset. Bins were generated with the new method for a bin size of 500kb and varying *Minimum Island Size* and *Bad Island Threshold*. The peak correlation is shown in bold. For the original 500kb bins, the correlation was 0.7408.

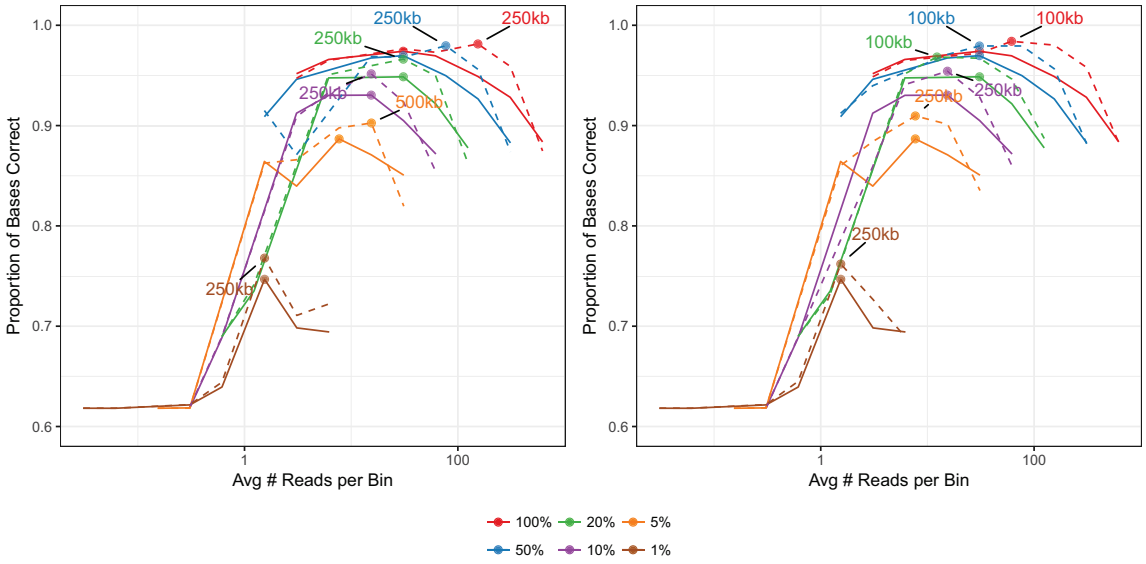


Figure 4.9: Comparison of CNV segmentation accuracy of the simulated dataset for original bins (solid) and new bins (dotted). New bins in the left figure were computed for a minimum island size of 0.2 and a bad island threshold of 0.7. New bins in the left figure were computed for a minimum island size of 0.3 and a bad island threshold of 0.5. The peak accuracy for each curve is labeled.

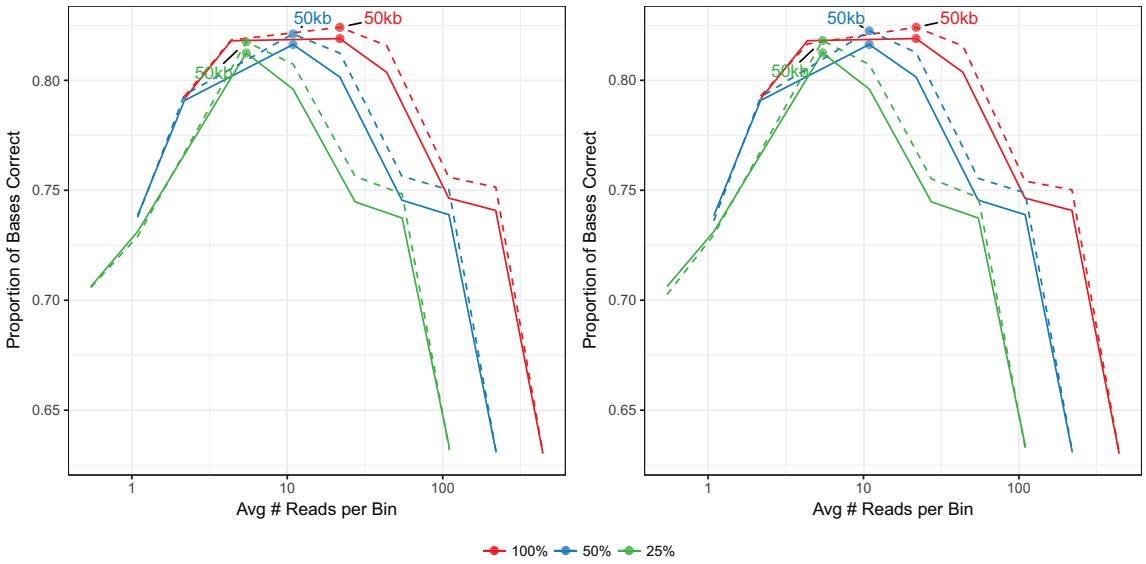


Figure 4.10: Comparison of SK-BR-3 CNV correlation for original bins (solid) and new bins (dotted). New bins in the left figure were computed for a minimum island size of 0.2 and a bad island threshold of 0.7. New bins in the left figure were computed for a minimum island size of 0.3 and a bad island threshold of 0.5. The peak correlation for each curve is labeled.

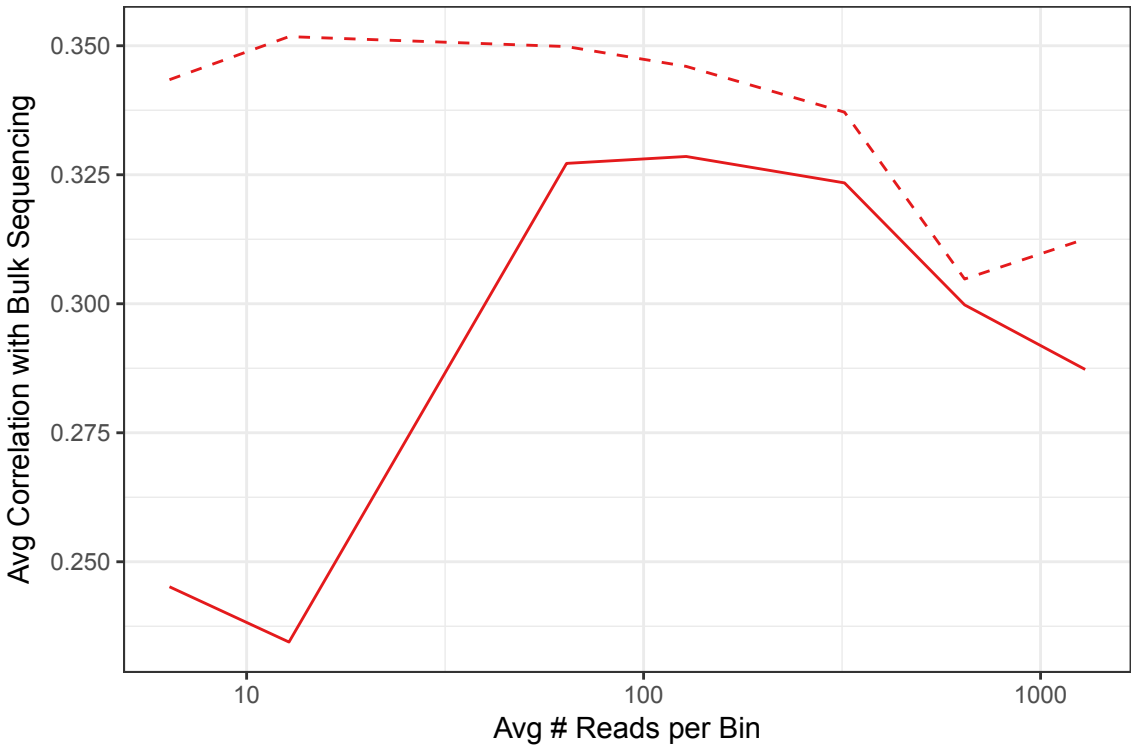


Figure 4.11: Comparison of MKN-45 CNV correlation for original bins (solid) and new bins (dotted). New bins were computed for a minimum island size of 0.2 and a bad island threshold of 0.6.

### 4.2.3 Bin Refinement

We expect our bin refinement method to be most useful when the initial Ginkgo segmentation must use large bins, for example due to lower read coverage. Refinement uses a fine bin size of 5kb, a 100x improvement in granularity over an initial bin size of 500kb. An initial bin size of 10kb, in contrast, would see little to no improvement from refinement with 5kb bins. Figures 4.12 through 4.14 show the increase in accuracy across multiple coverage levels and bin sizes when 5kb bins (the smallest bin available) are used for refinement. The improvement is most noticeable at the largest bin sizes and decreases to 0 as initial bin size decreases to 5kb since no further refinement is possible.

We also assessed the effect of bin refinement in SK-BR-3 cells on the copy number assigned to cancer-related genes, based on the COSMIC list of cancer-associated genes released by the Cancer Gene Consensus. For an initial bin size of 50kb refined to 5kb, we found that of 565 total genes, an average of 10.9 genes per cell changed copy number after bin refinement, including 5 genes that had a changed copy number state in at least 70 of the 94 cells.

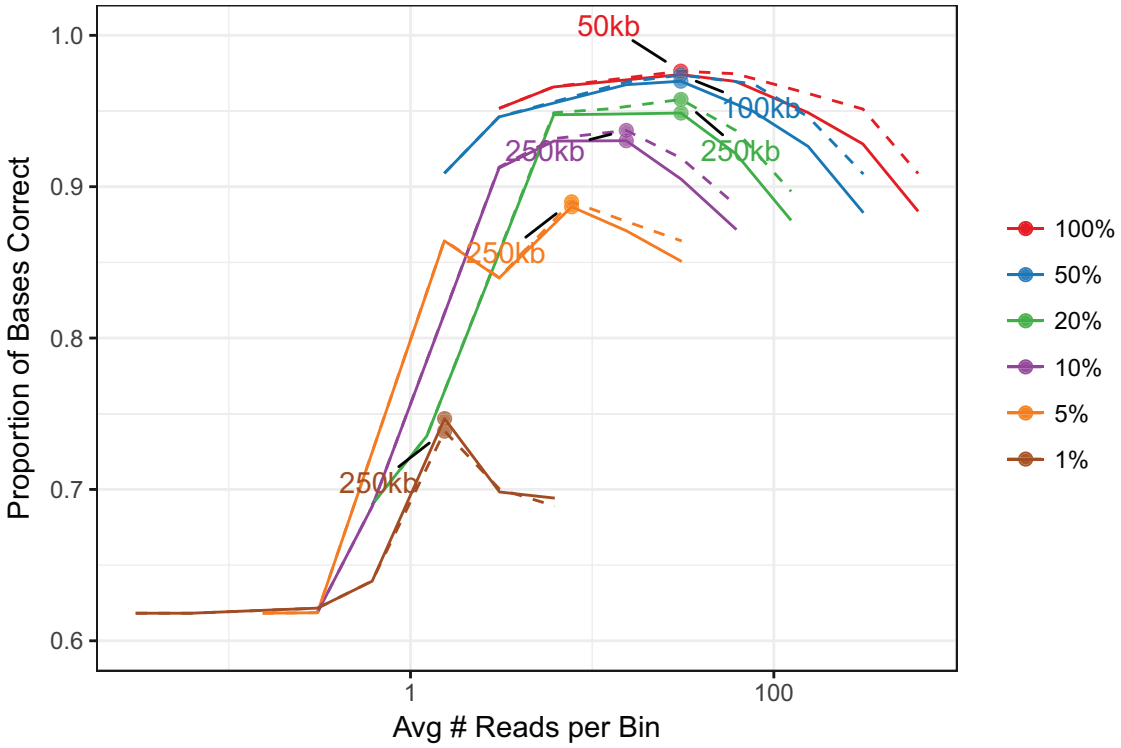


Figure 4.12: Accuracy improvement with bin refinement around copy-number changes. Each color shows a different downsampling of the original dataset and each curve runs from 1Mb bins (right) to 5kb (left). The peak accuracy for each curve is labeled.

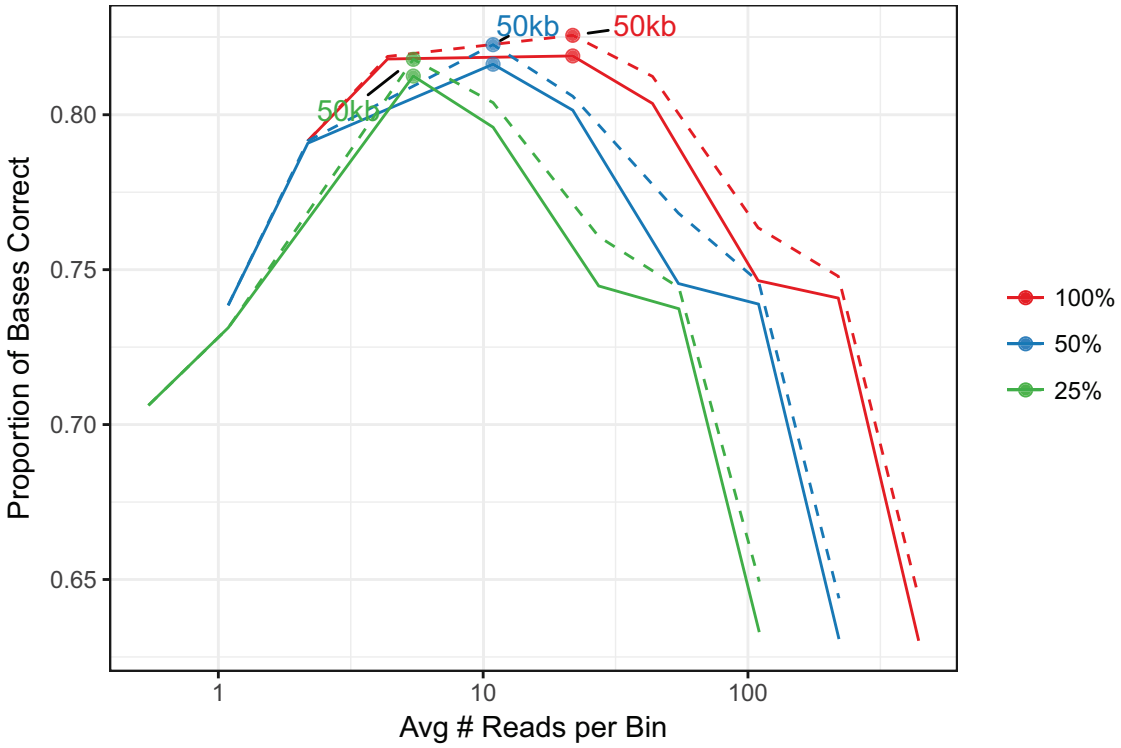


Figure 4.13: Improvement in SK-BR-3 CNV correlation after bin refinement around copy-number changes. Each color shows a different downsampling of the original dataset and each curve runs from 1Mb bins (right) to 5kb (left). The peak correlation for each curve is labeled.

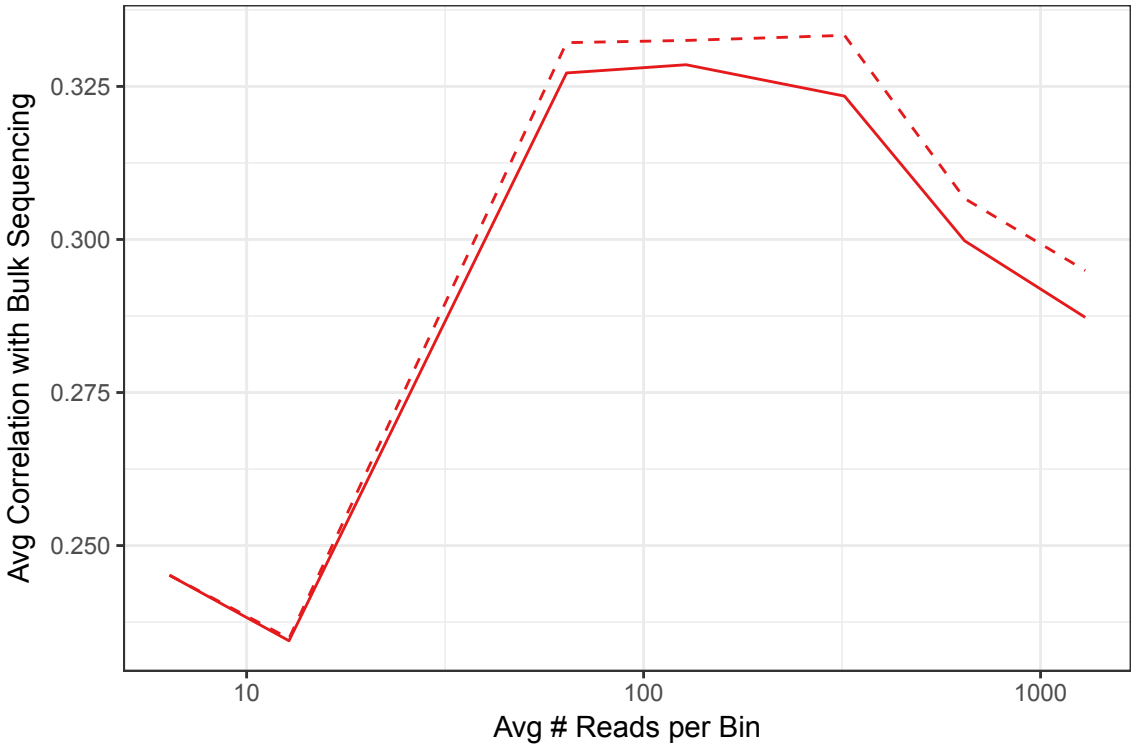


Figure 4.14: Improvement in MKN-45 CNV correlation after bin refinement around copy-number changes. Each color shows a different downsampling of the original dataset and each curve runs from 1Mb bins (right) to 5kb (left). The peak correlation for each curve is labeled.

### 4.2.4 Copy Number Results at Cancer-Associated Gene Loci

To validate the bin refinement method in a real-world setting, we analyzed copy numbers of the COSMIC list of cancer-associated genes released by the Cancer Gene Consensus for both the SK-BR-3 and MKN-45 cell lines. Figure 4.15 shows the absolute change in assigned copy number before and after bin refinement for an initial bin size of 50kb across all samples. Only those genes that changed in at least 3 samples are shown. Several genes show changes in a majority of samples, including the well known cancer genes CDH1, PBRM1, PTPRT, and TRIP11.

Figure 4.16 shows the absolute change in assigned copy number before and after bin refinement for an initial bin size of 50kb for a random subset of 200 samples. Only genes that changed in at least three samples are shown. Genes showing changes across a large proportion of samples include HIP1, MAML2, MAP2K4, NOTCH2, and PDE4DIP.

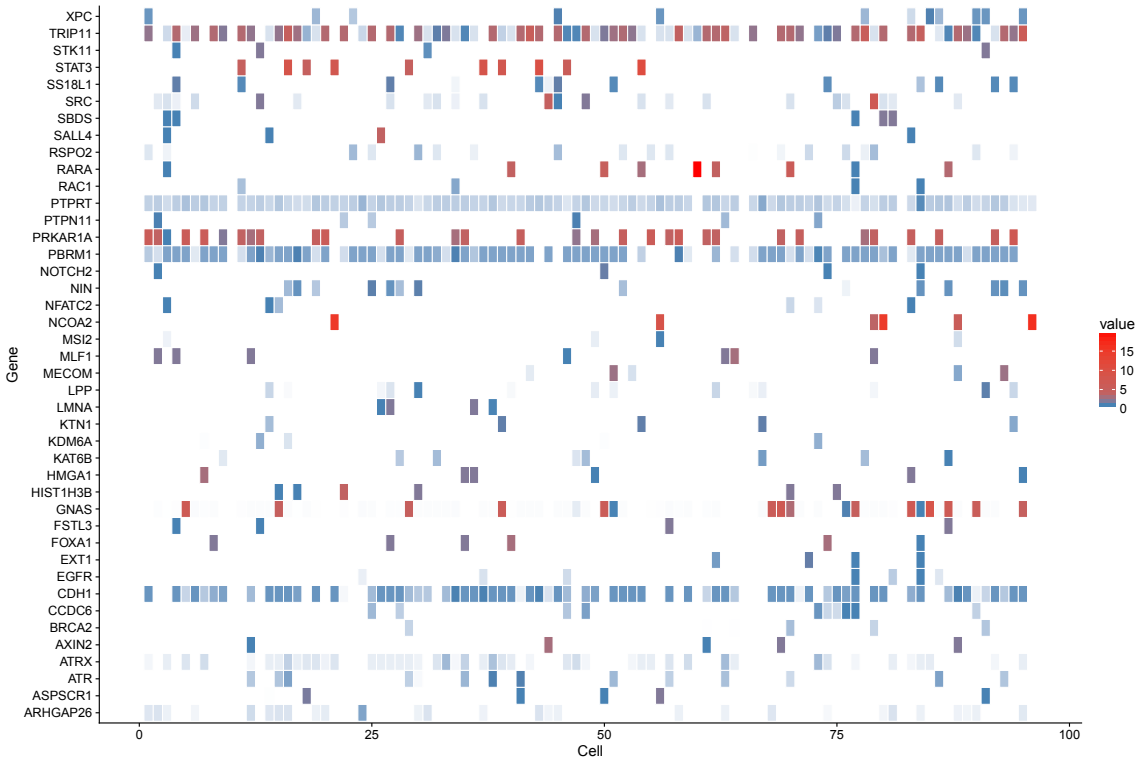


Figure 4.15: Heatmap of copy number changes of Cosmic cancer-associated genes in SK-BR-3 cells before and after bin refinement for an initial bin size of 50kb. We omitted genes for which copy number did not change in at least 3 samples.

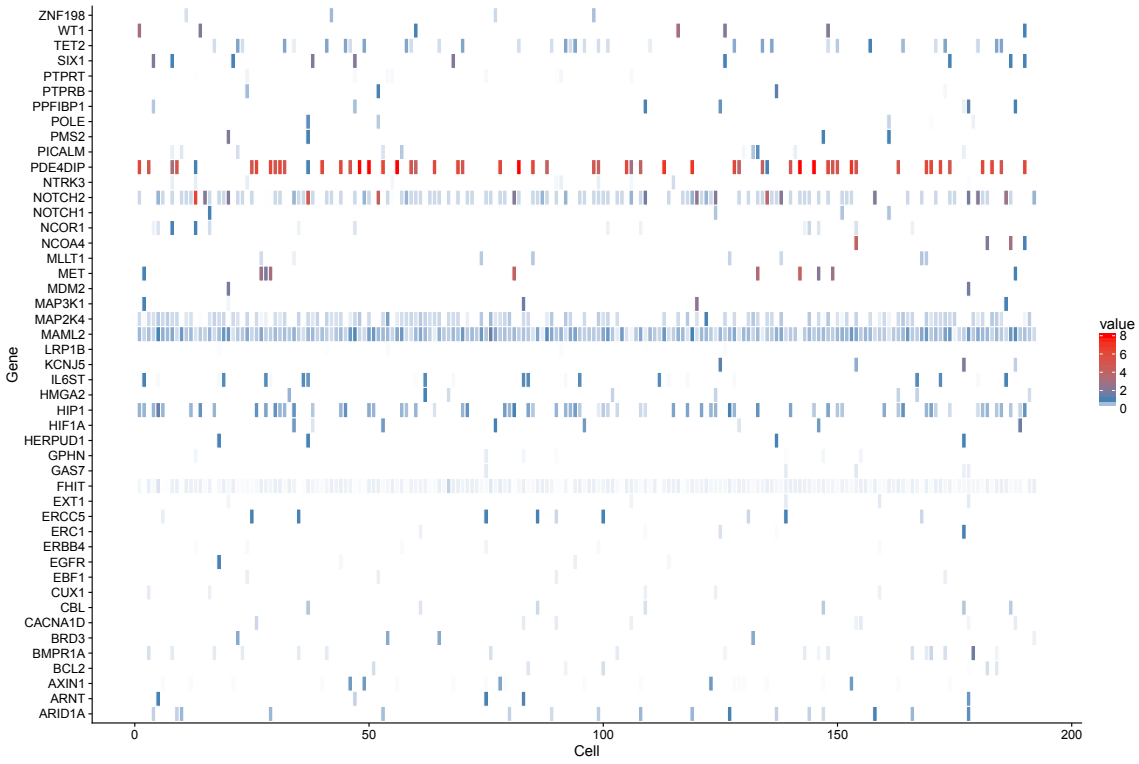


Figure 4.16: Heatmap of copy number changes of Cosmic cancer-associated genes in MKN-45 cells before and after bin refinement for an initial bin size of 50. We omitted genes for which copy number did not change in at least 3 samples.

## 4.3 Discussion

As single-cell sequencing becomes more efficient and widely adopted, researchers are applying it to increasingly diverse cell types and applications. In light of this recent boom, this manuscript has characterized some of the challenges of performing CNV analysis on single-cell data and presented methods to minimize their effect. Ginkgo 2 increases the accuracy of Ginkgo on low coverage data and improves scalability for large numbers of samples. Ginkgo 2 is available open source at `https://github.com/jpritt/ginkgodev` and a web platform is available at `https://qb.cshl.edu/ginkgodev`.

The key to accurate copy number analysis is the appropriate placement of bin boundaries and copy number segment boundaries. Boundary selection should be informed both by the genome sequence – taking repetitive sequences and other unmappable bases into account – and by the coverage profile of the dataset at hand. The methods described here address both these issues by improving the trade-off space between read coverage and bin size. When computing CNVs with Ginkgo it is important to choose an appropriate bin size. In general smaller bins lead to better precision, but in practice bin size is limited by the low coverage of samples.

The trade-off between bin size and read coverage affects CNV accuracy in at least 3 distinct ways. First, bin size and read coverage limits the detection of CNVs in the neighborhood of unmappable regions. Under the original binning implementation, larger bins are more likely to contain large regions containing a high density

of unmappable bases. In addition, bins in the neighborhood of unmappable bases tend to suffer from increased noise due to the unpredictability of those regions. Second, bin size affects the localization of changes in copy-number level. Ginkgo defines CNVs by the bins that they span; thus even if bins are classified correctly, CNV endpoints may be placed up half a bin away from their true location. Third, bin size and read coverage affect the detection of short CNVs. To avoid overfitting, Ginkgo accepts a parameter `minBinWidth` and only searches for CNVs that span a minimum of `minBinWidth` bins. This threshold is generally set to about 3–5, meaning that CNVs that span fewer than 3 bins are unlikely to be detected. Even if the threshold is set to 0, CNVs substantially shorter than the bin size will likely go undetected.

Each of the methods described in this chapter improves on the bin size - read coverage trade-off in one of these three areas. The new binning method improves segregation of highly-unmappable regions, thus improving CNV detection near those regions. Moreover, we can optimize improvement for a specific bin size by varying the parameters of the binning method. The bin refinement method improves the localization of CNV endpoints even when overall coverage is low. While bin refinement is still susceptible to local noise, this effect is diminished by first segmenting with a large bin size, and use the smallest bins only in highly localized regions. Figure 4.12 demonstrates that bin refinement corrects some of the errors due to low coverage; specifically, the 50%-downsampled dataset achieves nearly the same peak accuracy as the full dataset. Finally, aggregation of technical replicates effectively increases read

coverage, improving accuracy in all three areas. Particularly, it offers the opportunity to detect small CNVs that were previously engulfed by large bins.

# Chapter 5

# Future Directions

The tools and research questions discussed in this thesis present many promising extensions. I discuss several of the most interesting below.

### Boiler

Boiler presents many opportunities for innovation, including extension to multi-mapped reads and multi-sample datasets, and integration with existing aligners and downstream tools.

One direction that I find particularly interesting is the ability to choose the amount of lossy-ness introduced by compression. The current Boiler algorithm is an extremely lossy but efficient compression method. In contrast, tools like Goby or CRAM present a lossless but less compressible alternative. One might imagine a lossy-ness slider between these two extremes, set by the user for their particular application.

Although Boiler's current method represents an extreme level of lossy-ness, intermediate amounts might be introduced by storing some additional information alongside each bucket. For example, storing all read starting positions would improve read localization, but would still leave some ambiguity in read lengths. While this example is a binary option, a true "sliding-scale" feature would require development and optimization of a continuous or fine-grained variable that can be adjusted from lossless to fully lossy.

**FORGe**

While I have demonstrated that an optimal subset of variants exists for the graph genome, I presented only two ranking methods here, one of which is not yet tractable for the full human genome. Extension of the hybrid ranking to the full human genome requires use of an approximate k-mer counter and is not yet complete. In the future, I hope to explore alternative ranking strategies, such as a haplotype-centric pseudocontig ranking (in contrast to the methods presented here which treat variants independently.)

I would also like to extend the methods presented here to other organisms; plant genomes such as *Arabidopsis thaliana* might particularly benefit from graph genome analysis.

### Ginkgo

Single-cell sequencing is still a relatively novel technology, even within the context of genomic sequencing. As a result, the full range of applications has yet to be fully explored. As single-cell sequencing becomes increasingly available, the demands placed on analysis tools will continue to rise and I anticipate that researchers will require yet unlooked-for features.

One such feature is association of a score with each copy-number value indicating Ginkgo's confidence in the correctness of that assignment, similar to the "mapping quality" reported by aligners. Confidence scores would make CNV calls more meaningful and enable more accurate interpretation. In order to truly be useful, confidence scores must be computed from a statistical basis and should reflect such values as (1) the probability computed by the segmentation algorithm, (2) the variability of bin counts, or noisiness of the data, and (3) confidence in the chosen ploidy. Formulation of a statistical basis would be the first step toward assigning confidence scores to CNVs.

# Bibliography

[1] International Human Genome Sequencing Consortium. "Initial sequencing and analysis of the human genome". In: *Nature* 409 (6822 2001), pp. 860–921.

[2] Venter JC et al. "The Sequence of the Human Genome". In: *Science* 291 (5507 2001), pp. 1304–1351.

[3] Shendure J, Balasubramanian S, Church GM, Gilbert W, Rogers J, Schloss JA, and Waterston RH. "DNA sequencing at 40: past, present and future". In: *Nature* 550 (7676 2017), pp. 345–353.

[4] Langmead B and Nellore A. "Cloud computing for genomic data analysis and collaboration". In: *Nature Reviews Genetics* 19 (2018), pp. 208–219.

[5] K. A. Frazer, D. G. Ballinger, D. R. Cox, D. A. Hinds, L. L. Stuve, R. A. Gibbs, J. W. Belmont, A. Boudreau, P. Hardenbol, S. M. Leal, et al. "A second generation human haplotype map of over 3.1 million SNPs". In: *Nature* 449.7164 (Oct. 2007), pp. 851–861.

[6] A. Auton, L. D. Brooks, R. M. Durbin, E. P. Garrison, H. M. Kang, J. O. Korbel, J. L. Marchini, S. McCarthy, G. A. McVean, G. R. Abecasis, et al. "A global reference for human genetic variation". In: *Nature* 526.7571 (Oct. 2015), pp. 68–74.

[7] K. Walter, J. L. Min, J. Huang, L. Crooks, Y. Memari, S. McCarthy, J. R. Perry, C. Xu, M. Futema, D. Lawson, et al. "The UK10K project identifies rare variants in health and disease". In: *Nature* 526.7571 (Oct. 2015), pp. 82–90.

[8] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biol.* 10.3 (2009), R25.

[9] Li H and Durbin R. "Fast and accurate short read alignment with Burrows-Wheeler Transform". In: *Bioinformatics* 25 (2009), pp. 1754–1760.

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

[11] Patro R, Mount SM, and Kingsford C. "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms". In: *Nature Biotechnology* 32 (5 2014), pp. 462–464.

[12] Bray NL, Pimentel H, Melsted P, and Pachter L. "Near-optimal probabilistic RNA-seq quantification". In: *Nature Biotechnology* 34 (5 2016), pp. 525–527.

[13] Patro R, Duggal G, Love MI, Irizarry RA, and Kingsford C. "Salmon provides fast and bias-aware quantification of transcript expression". In: *Nature Methods* 14 (4 2017), pp. 417–419.

[14] Sugawara H Leinonen R and Shumway M. "The sequence read archive". In: *Nucleic acids research* 39 (suppl 1 2011), pp. D19–D21.

[15] Lappalainen T, Sammeth M, Friedländer MR, t Hoen P, Monlong J, Rivas MA, Gonzalez-Porta M, Kurbatova N, Griebel T, and Ferreira PG et al. "Transcriptome and genome sequencing uncovers functional variation in humans". In: *Nature* 501 (7468 2013), pp. 506–511.

[16] Ardlie KG et al. "The Genotype-Tissue Expression (GTEx) pilot analysis: Multitissue gene regulation in humans". In: *Science* 348 (6235 2015), pp. 648–660.

[17] Pritt J and Langmead B. "Boiler: lossy compression of RNA-seq alignments using coverage vectors". In: *Nucleic Acids Research* 44 (16 2016), e133.

[18] The Computational Pan-Genomics Consortium. "Computational pan-genomics: status, promises and challenges". In: *Brief. Bioinformatics* (Oct. 2016).

[19] B. Paten, A. M. Novak, J. M. Eizenga, and E. Garrison. "Genome graphs and the evolution of genome inference". In: *Genome Res.* 27.5 (May 2017), pp. 665–676.

[20] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel. "Simultaneous alignment of short reads against multiple genomes". In: *Genome Biol.* 10.9 (2009), R98.

[21] R. V. Satya, N. Zavaljevski, and J. Reifman. "A new strategy to reduce allelic bias in RNA-Seq readmapping". In: *Nucleic Acids Res.* 40.16 (Sept. 2012), e127.

[22] L. Huang, V. Popic, and S. Batzoglou. "Short read alignment with populations of genomes". In: *Bioinformatics* 29.13 (July 2013), pp. i361–370.

[23] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. "Indexing graphs for path queries with applications in genome research". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11.2 (2014), pp. 375–388.

[24] Jouni Sirén. "Indexing Variation Graphs". In: *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2017, pp. 13–27.

[25] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Michael F Lin, Benedict Paten, and Richard Durbin. "Sequence variation aware genome references and read mapping with the variation graph toolkit". In: *bioRxiv* (2017). DOI: 10.1101/234856. eprint: https://www.biorxiv.org/content/early/2017/12/15/234856.full.pdf. URL: https://www.biorxiv.org/content/early/2017/12/15/234856.

[26] D. Kim, J.M. Paggi, and S.L. Salzberg. "HISAT-genotype: Next Generation Genomic Analysis Platform on a Personal Computer". In: *bioRxiv* (2018). DOI: 10.1101/266197.

[27] S. Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. "A natural encoding of genetic variation in a Burrows-Wheeler Transform to enable mapping and genome inference". In: *International Workshop on Algorithms in Bioinformatics*. Springer. 2016, pp. 222–233.

[28] Pritt J and Langmead B. "FORGe: prioritizing variants for graph genomes". In: *bioRxiv* (2018). DOI: 10.1101/311720.

[29] Biezuner T Shapiro E and Linnarsson S. "Single-cell sequencing-based technologies will revolutionize whole-organism science". In: *Nature Reviews Genetics* 14 (9 2013), pp. 618–630. DOI: 10.1038/nrg3542.

[30] Navin N. "The first five years of single-cell cancer genomics and beyond". In: *Genome Research* 25 (2015), pp. 1499–1507.

[31] McConnell MJ et al. "Mosaic Copy Number Variation in Human Neurons". In: *Science* 342 (6158 2013), pp. 632–637. DOI: 10.1126/science.1243472.

[32] Trapnell C. "Defining cell types and states with single-cell genomics". In: *Genome Research* 25 (2015), pp. 1491–1498.

[33] Navin N et al. "Tumour evolution inferred by single-cell sequencing". In: *Nature* 472 (2011), pp. 90–94. DOI: 10.1038/nature09807.

[34] Wang Y et al. “Clonal evolution in breast cancer revealed by single nucleus genome sequencing”. In: *Nature* 512 (2014), pp. 155–160.

[35] Ni X et al. “Reproducible copy number variation patterns among single circulating tumor cells of lung cancer patients”. In: *PNAS* 110 (2013), pp. 21083–21088.

[36] Behr B Wang J Fan HC and Quake SR. “Genome-Wide Single-Cell Analysis of Recombination Activity and de novo Mutation Rates in Human Sperm”. In: *Cell* 150 (2 2012), pp. 402–412. DOI: 10.1016/j.cell.2012.06.030.

[37] Henrichsen C, Chaignat E, and Reymond A. “Copy number variants, diseases and gene expression”. In: *Human Molecular Genetics* 18 (2009), R1–R8.

[38] Baslan T et al. “Genome-wide copy number analysis of single cells”. In: *Nature Protocols* 7 (2012), pp. 1024–1041.

[39] Kendall J et al Garvin T Aboukhalil R. “Interactive analysis and assessment of single-cell copy-number variations”. In: *Nature methods* 12 (11 2015), pp. 1058–1060. DOI: 10.1038/nmeth.3578.

[40] Cochrane G Hsi-Yang FM Leinonen R and Birney E. “Efficient storage of high throughput DNA sequencing data using reference-based compression”. In: *Genome Research* 5 (2011), pp. 734–740.

[41] Popitsch N and von Haeseler A. “NGC: lossless and lossy compression of aligned high-throughput sequencing data”. In: *Nucleic Acids Research* 41 (1 2013), e27.

[42] Campagne F et al. “Compression of Structured High-Throughput Sequencing Data”. In: *PLoS ONE* 8 (11 2013), e79871.

[43] Filippova D and Kingsford C. “Rapid, separable compression enables fast analyses of sequence alignments”. In: *ACM Conference on Bioinformatics* (2015), pp. 194–201.

[44] Daily K, Rigor P, Christley S, Xie X, and Baldi P. “Data structures and compression algorithms for high-throughput sequencing technologies”. In: *BMC Bioinformatics* 11 (1 2010), p. 514.

[45] Kozanitis C, Saunders C, Kruglyak S, Bafna V, and Varghese G. “Compressing genomic sequence fragments using SlimGene”. In: *Journal of Computational Biology* 18 (3 2011), pp. 401–413.

[46] Peng X Jones DC Ruzzo WL and Katze MG. “Compression of next-generation sequencing reads aided by highly efficient de novo assembly”. In: *Nucleic Acids Research* 40 (22 2012), e171.

[47] Trapnell C, Williams B, Pertea G, Mortazavi A, Kwan G, van Baren J, Salzberg S, Wold B, and Pachter L. In: *Nature Biotechnology* 28 (5 2010), pp. 511–515.

[48] Pertea M, Pertea GM, Antonescu CM, Chang TS, Mendell JT, and Salzberg SL. “StringTie enables improved reconstruction of a transcriptome from RNA-seq reads”. In: *Nature Biotechnology* 33 (3 2015), pp. 290–295.

[49] Langmead B Kim D and Salzberg SL. "HISAT: a fast spliced aligner with low memory requirements". In: *Nature Methods* 12 (4 2013), pp. 357–360.

[50] Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, and Salzberg SL. "TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions". In: *Genome Biology* 14 (4 2013), R36.

[51] Caprara A, Kellerer H, and Pferschy U. "A PTAS for the multiple subset sum problem with different knapsack capacities". In: *Information Processing Letters* 73 (3 2000), pp. 111–118.

[52] Garey MR and Johnson DS. ""Strong" NP-Completeness Results: Motivation, Examples, and Implications". In: *Journal of the ACM* 25 (3 1978), pp. 499–508.

[53] Griebel T, Zacher B, Ribeca P, Raineri E, Lacroix V, Guigó R, and Sammeth M. "Modelling and simulating generic RNA-Seq experiments with the flux simulator". In: *Nucleic acids research* 40 (20 2012), pp. 10073–10083.

[54] Yates A, Akanni W, Amode MR, Barrell D, Billis K, Carvalho-Silva D, Cummins C, Clapham P, Fitzgerald S, and Gil L et al. "Ensembl 2016". In: *Nucleic Acids Research* 28 (5 2015), gkv1157.

[55] Harrow J, Frankish A, Gonzalez JM, Tapanari E, Diekhans M, Kokocinski F, Aken BL, Barrell D, Zadissa A, and Searle S et al. "GENCODE: the reference human genome annotation for The ENCODE Project". In: *Genome Research* 22 (9 2012), pp. 1760–1774.

[56] SEQC/MAQC-III Consortium. "A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium". In: *Nature Biotechnology* 32 (9 2014), pp. 903–914.

[57] Li B, Fillmore N, Bai Y, Collins M, Thomson J, Stewart R, and Dewey C. "Evaluation of de novo transcriptome assemblies from RNA-Seq data". In: *Genome Biology* 15 (12 2014), p. 553.

[58] Smyth GK Liao Y and Shi W. "featureCounts: an efficient general purpose program for assigning sequence reads to genomic features". In: *Bioinformatics* 30 (7 2014), pp. 923–930.

[59] Pyl PT Anders S and Huber W. "HTSeq–A Python framework to work with high-throughput sequencing data". In: *Bioinformatics* 31 (2 2014), pp. 166–169.

[60] Frazee A, Sabunciyan S, Hansen K, Irizarry R, and Leek J. "Differential expression analysis of RNA-seq data at single-base resolution". In: *Biostatistics* 15 (3 2014), pp. 413–426.

[61] Heng Li, Jue Ruan, and Richard Durbin. "Mapping short DNA sequencing reads and calling variants using mapping quality scores". In: *Genome research* 18.11 (2008), pp. 1851–1858.

[62] B. Langmead. "A tandem simulation framework for predicting mapping quality". In: *Genome Biol.* 18.1 (Aug. 2017), p. 152.

[63] A. Dilthey, C. Cox, Z. Iqbal, M. R. Nelson, and G. McVean. "Improved genome inference in the MHC using a population reference graph". In: *Nat. Genet.* 47.6 (June 2015), pp. 682–688.

[64] X. Gan, O. Stegle, J. Behr, J. G. Steffen, P. Drewe, K. L. Hildebrand, R. Lyngsoe, S. J. Schultheiss, E. J. Osborne, V. T. Sreedharan, et al. "Multiple reference genomes and transcriptomes for Arabidopsis thaliana". In: *Nature* 477.7365 (Aug. 2011), pp. 419–423.

[65] J. F. Degner, J. C. Marioni, A. A. Pai, J. K. Pickrell, E. Nkadori, Y. Gilad, and J. K. Pritchard. "Effect of read-mapping biases on detecting allele-specific expression from RNA-sequencing data". In: *Bioinformatics* 25.24 (Dec. 2009), pp. 3207–3212.

[66] D. Y. Brandt, V. R. Aguiar, B. D. Bitarello, K. Nunes, J. Goudet, and D. Meyer. "Mapping Bias Overestimates Reference Allele Frequencies at the HLA Genes in the 1000 Genomes Project Phase I Data". In: *G3 (Bethesda)* 5.5 (Mar. 2015), pp. 931–941.

[67] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, et al. "A framework for variation discovery and genotyping using next-generation DNA sequencing data". In: *Nat. Genet.* 43.5 (May 2011), pp. 491–498.

[68] H. P. Eggertsson, H. Jonsson, S. Kristmundsdottir, E. Hjartarson, B. Kehr, G. Masson, F. Zink, K. E. Hjorleifsson, A. Jonasdottir, A. Jonasdottir, et al. "Graphtyper enables population-scale genotyping using pangenome graphs". In: *Nat. Genet.* 49.11 (Nov. 2017), pp. 1654–1660.

[69] B. Liu, H. Guo, M. Brudno, and Y. Wang. "deBGA: read alignment with de Bruijn graph-based seed and extension". In: *Bioinformatics* 32.21 (Nov. 2016), pp. 3224–3232.

[70] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. "De novo assembly and genotyping of variants using colored de Bruijn graphs". In: *Nat. Genet.* 44.2 (Jan. 2012), pp. 226–232.

[71] Paolo Ferragina and Giovanni Manzini. "Opportunistic data structures with applications". In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE. 2000, pp. 390–398.

[72] A. Danek, S. Deorowicz, and S. Grabowski. "Indexes of large genome collections on a PC". In: *PLoS ONE* 9.10 (2014), e109384.

[73] D. M. Church, V. A. Schneider, T. Graves, K. Auger, F. Cunningham, N. Bouk, H. C. Chen, R. Agarwala, W. M. McLaren, G. R. Ritchie, et al. "Modernizing reference genome assemblies". In: *PLoS Biol.* 9.7 (July 2011), e1001091.

[74] D. M. Church, V. A. Schneider, K. M. Steinberg, M. C. Schatz, A. R. Quinlan, C. S. Chin, P. A. Kitts, B. Aken, G. T. Marth, M. M. Hoffman, et al. "Extending reference assembly models". In: *Genome Biol.* 16 (Jan. 2015), p. 13.

[75] F. E. Dewey, R. Chen, S. P. Cordero, K. E. Ormond, C. Caleshu, K. J. Karczewski, M. Whirl-Carrillo, M. T. Wheeler, J. T. Dudley, J. K. Byrnes, et al. "Phased whole-genome genetic risk in a family quartet using a major allele reference sequence". In: *PLoS Genet.* 7.9 (Sept. 2011), e1002280.

[76] S. Karthikeyan, P. S. Bawa, and S. Srinivasan. "hg19K: addressing a significant lacuna in hg19-based variant calling". In: *Mol Genet Genomic Med* 5.1 (Jan. 2017), pp. 15–20.

[77] S. T. Sherry, M. H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin. "dbSNP: the NCBI database of genetic variation". In: *Nucleic Acids Res.* 29.1 (Jan. 2001), pp. 308–311.

[78] G. Marcais and C. Kingsford. "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers". In: *Bioinformatics* 27.6 (Mar. 2011), pp. 764–770.

[79] Manuel Holtgrewe. "Mason–a read simulator for second generation sequencing data". In: *Technical Report FU Berlin* (2010).

[80] I. Dolgalev, F. Sedlazeck, and B. Busby. "DangerTrack: A scoring system to detect difficult-to-assess regions". In: *F1000Res* 6 (2017), p. 443.

[81] H. Thorvaldsdottir, J. T. Robinson, and J. P. Mesirov. "Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration". In: *Brief. Bioinformatics* 14.2 (Mar. 2013), pp. 178–192.

[82] L. Maretty, J. M. Jensen, B. Petersen, J. A. Sibbesen, S. Liu, P. Villesen, L. Skov, K. Belling, C. Theil Have, J. M. G. Izarzugaza, et al. "Sequencing and de novo assembly of 150 genomes from Denmark as a population reference". In: *Nature* 548.7665 (Aug. 2017), pp. 87–91.

[83] Adam Ameur, Huiwen Che, Marcel Martin, Ignas Bunikis, Johan Dahlberg, Ida Höijer, Susana Häggqvist, Francesco Vezzi, Jessica Nordlund, Pall Olason, et al. "De novo assembly of two Swedish genomes reveals missing segments from the human GRCh38 reference and improves variant calling of population-scale sequencing data". In: *bioRxiv* (2018). DOI: 10.1101/267062. eprint: https://www.biorxiv.org/content/early/2018/02/18/267062.full.pdf. URL: https://www.biorxiv.org/content/early/2018/02/18/267062.

[84] L. Shi, Y. Guo, C. Dong, J. Huddleston, H. Yang, X. Han, A. Fu, Q. Li, N. Li, S. Gong, et al. "Long-read sequencing and de novo assembly of a Chinese genome". In: *Nat Commun* 7 (June 2016), p. 12065.

[85] M. A. Eberle, E. Fritzilas, P. Krusche, M. Kallberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H. Y. Chuang, S. J. Humphray, A. L. Halpern, et al. "A reference data set of 5.4 million phased human variants validated by genetic inheritance

from sequencing a three-generation 17-member pedigree". In: *Genome Res.* 27.1 (Jan. 2017), pp. 157–164.

[86] A. M. Novak, G. Hickey, E. Garrison, S. Blum, A. Connelly, A. Dilthey, J. Eizenga, M. A. S. Elmohamed, S. Guthrie, A. Kahles, et al. "Genome Graphs". In: *BioRxiv* (2017).

[87] H. Lee and C. Kingsford. "Kourami: graph-guided assembly for novel human leukocyte antigen allele discovery". In: *Genome Biol.* 19.1 (Feb. 2018), p. 16.

[88] T. D. Otto, M. Sanders, M. Berriman, and C. Newbold. "Iterative Correction of Reference Nucleotides (iCORN) using second generation sequencing technology". In: *Bioinformatics* 26.14 (July 2010), pp. 1704–1707.

[89] G. Genovese, R. E. Handsaker, H. Li, N. Altemose, A. M. Lindgren, K. Chambert, B. Pasaniuc, A. L. Price, D. Reich, C. C. Morton, et al. "Using population admixture to help complete maps of the human genome". In: *Nat. Genet.* 45.4 (Apr. 2013), pp. 406–414.

[90] J. S. Seo, A. Rhie, J. Kim, S. Lee, M. H. Sohn, C. U. Kim, A. Hastie, H. Cao, J. Y. Yun, J. Kim, et al. "De novo assembly and phasing of a Korean human genome". In: *Nature* 538.7624 (Oct. 2016), pp. 243–247.

[91] M. J. Chaisson, J. Huddleston, M. Y. Dennis, P. H. Sudmant, M. Malig, F. Hormozdiari, F. Antonacci, U. Surti, R. Sandstrom, M. Boitano, et al. "Resolv-

ing the complexity of the human genome using single-molecule sequencing". In: *Nature* 517.7536 (Jan. 2015), pp. 608–611.

[92] Venkatraman E. Seshan and Adam Olshen. *DNAcopy: DNA copy number data analysis*. R package version 1.46.0. 2016.

[93] Futreal P et al. "A census of human cancer genes". In: *Nature Reviews Cancer* 4 (2004), pp. 177–183.

[94] Yoon K et al. "Comprehensive genome- and transcriptome-wide analyses of mutations associated with microsatellite instability in Korean gastric cancers". In: *Genome Research* (23 2013), pp. 1109–1117.

# Vita

M. Jacob Pritt was born in 1991.

Jacob did his undergraduate work at Havard University, where he majored in Comuter Science and minored in Molecular and Cellular Biology. He spent summers as an intern in computational biology labs at the National Institues of Health and at the Wyss Institute. Jacob worked as a teaching assistant for several computer science classes including Introduction to Computer Science and Visualization.

In 2013, Jacob began his PhD at Johns Hopkins University. He was a teaching assistant for Ben Langmead's Computational Genomics class.