UNIVERSITY OF CALIFORNIA
SANTA CRUZ

GRAPH METHODS FOR COMPUTATIONAL PANGENOMICS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

BIOINFORMATICS

by

Jordan M. Eizenga

September 2021

The Dissertation of Jordan M. Eizenga
is approved:

Professor Benedict Paten, Chair

Professor David Haussler

Professor R. Edward Green

Peter F. Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Graph methods for computational pangenomics

by

Jordan M. Eizenga

In most sequencing experiments, sequencing reads are mapped to a reference genome assembly in order to identify the genomic elements that the reads originated from. The mapping process becomes less accurate when the sample's genome differs from the reference genome. This introduces a pervasive *reference bias* in which genomics analyses are systematically less accurate for non-reference alleles. In the field of pangenomics, it has been proposed that more general reference structures could mitigate reference bias. The fundamental idea is to incorporate population variation into the reference itself. The result is naturally expressible as a sequence graph. This dissertation presents the research I performed to develop methods for graph-based pangenomic analyses. First, I describe a read mapping and inference pipeline to perform haplotype-resolve transcriptomic analyses using pangenomics techniques. Next, I describe several contributions I have made to the ecosystem of pangenomic software: an interface to conventional reference methods, a software library of pangenome graph data structures, and a usable interface for indexing pangenome graphs. Finally, I describe some applications of graph theory to pangenome graphs to perform practical pangenomics tasks: identifying sites of variation and converting overlapped sequence graphs to blunt ones.

To my parents,

Douglas and Lori Eizenga,

who I can always count on,

and to my dear friend,

Corrie Janssens,

my moral North Star

Acknowledgments

I would like to thank the members of my committee, Benedict Paten, David Haussler, and Ed Green for their insight and critique of the work presented in my dissertation and defense. I am particularly grateful to Benedict Paten for the years of mentorship he has provided me in the Computational Genomics Lab. His guidance taught me how to conduct science and led me to a challenging and rewarding area of study.

My lab mates have been my collaborators, commiserators, and friends. I am grateful to have worked alongside Art Rand, Miten Jain, Joel Armstrong, Ian Fiddes, Arjun Rao, Adam Novak, Glenn Hickey, John Vivian, Audrey Musselman-Brown, Trevor Peasout, Molly Zhang, Yohei Rosen, Xian Chang, Jouni Sirén, Marina Haukness, Kishwar Shafin, Andrew Bailey, Colleen Bosworth, Ryan Lorig-Roach, Jean Monlong, Jonas Sibbesen, Robin Rounthwaite, James Casaletto, Yatish Turakhia, Melissa Meredith, Cecilia Cisar, and Mobin Asri. For the last six years, they have provided me with an intellectually stimulating environment and with a sense of community. Outside of my lab, I feel I should also specifically thank my friend and collaborator Erik Garrison, whose vision and leadership were motive forces in the research community I belong to.

The route I took into bioinformatics was rather circuitous. In my mind, the most crucial determinative period was my mid-twenties, when I was broke, depressed, and unable to work after developing a chronic medical condition. I ended up getting the lifeline I needed from the burgeoning availability of free online college-level educational materials. Engaging in these courses pulled me out of that slump and completely reshaped my interests, which is

ultimately what led me to graduate school in bioinformatics. Because of this, I feel grateful to a number of professors I have never personally met, and who yet profoundly impacted my life. In particular, I would like to thank David J. Malan, Eric Lander, Pavel Pevzner, Phillip Compeau, Tim Roughgarden, Leonard E. White, Andrew Ng, Marnie Blewitt, Rob Tibshirani, Trevor Hastie, Eric Grimson, and Alma Novotny for their generosity with their time and expertise.

The University of California Santa Cruz has been my home for six years, and I am grateful to the people in the campus community who are working to make it a fairer and juster place. I want to thank the COLA wildcats for the risks they took and the sacrifices they made on behalf of all UCSC graduate students. I would particularly like to thank Carlos Cruz who continues to face unjust retaliation for his participation in the movement. The mobilization that the wildcats galvanized has borne further fruit in the Student Researchers United campaign, which I am proud to have been a part of. I am grateful for my fellow members of the organizing committee, as well as the 1000+ graduate students who have contributed to the campaign across the UC system.

Many more people have made the last six years of my life feel rich and meaningful. I am grateful for my community of friends, both in Santa Cruz and Michigan; I am grateful for my students, whose curiosity reinvigorates my own interest; I am grateful for and proud of the undergraduate researchers I have had the pleasure of mentoring; and I am grateful for the administrative staff who shepherded me through the whole process.

Finally and most of all, I would like to thank the members of my family. My mom and dad have supported my love of learning for my entire life. They have demonstrated over and over again that if I ever slip, someone will always be there to catch me. My brother Quentin's

relentless enthusiasm and dedication inspired me to take the plunge into graduate school in the first place. My brother Nate is a dependable match to my own nerdiness and sense of humor. My dear friend Corrie, who I consider family, is my personal hero, and I am continually impressed with her wisdom and growth.

In sum, this thesis is very much a group project. In most group projects, one student does all the work, and then the entire group shares the benefit. The situation here is the inverse. I highly doubt I will be allowed to share my doctorate with my collaborators and supporters, but I hope they understand that in a very real sense it is theirs too.

Part I

Introduction and background

Chapter 1

Introduction

Modern biologists can ask and answer a qualitatively greater set of questions at a quantitatively greater scale than their predecessors. Much of this expansion is reflected in the growing use of so-called *omics* methods. This term cheekily refers to a variety of methods that attempt to characterize the entirety of some class of biomolecular entity in a sample: all DNA in genomics, all proteins in proteomics, all metabolites in metabolomics, etc. Earlier methods in molecular biology that focused on individual entities (e.g. one gene) sometimes suffered a tunnel vision that could seriously slant the scientific understanding of a biological process. The widened lens of omics methods alleviates this risk, and thus they are often referred to as *unbiased* methods (although it may be more accurate to call them less-biased methods—more on this soon).

Undergirding the growth of omics methods is a deepened arsenal of technologies, and DNA sequencing is undoubtedly one of the most fundamental among them. The centrality of sequencing is due in part to the fundamental importance of DNA in biology; DNA is the (nearly)

universal medium of heredity and the primary object of evolutionary change. In addition, DNA sequencing has proven to be highly adaptable for assaying various biological processes. Many unbiased assays consist mainly in a method to imprint the effects of some process onto DNA molecules, which is then followed by sequencing.

The vast majority of sequencing experiments involve *resequencing*: generating sequencing data from an organism for which some individual's genome has been previously sequenced and assembled. For humans and model species, the assembled genomes are maintained as high-quality public data resources, referred to as *reference genomes*. In practice, this turns out to be crucial for two interrelated reasons. First, the reference serves as a technical tool to identify the genomic element corresponding to raw sequencing data. Second, the reference coordinate system acts as a proxy for homology between individual haplotypes. Together, these features allow sequencing experiments to be interpreted and generalized beyond the individual sample so that they can contribute to scientific understanding of biology.

Reference-based analysis is an expedient and well-developed methodology, but it does have downsides. For one, there is a fundamental limitation in using a single genome as a reference for an entire species. At best, a reference assembly can accurately represent the genomic sequence of a single haplotype of a single individual. Every other individual will have genetic differences so that their genome mismatches the reference in many locations. This introduces some ambiguity into the reference's role as a coordinate system. More subtly, it also diminishes the reference's ability to serve as a technical tool. The way that sequencing data is identified with a genomic element in the reference is by looking for sequences in the reference that match the data. If in fact the reference mismatches the genome that generated the data, it is

more difficult to identify the correct element, and the accuracy of this process deteriorates. This problem is known as *reference bias*: the systematic tendency for analyses to reduce in accuracy when a sample genome sequence differs from the reference.

Recently, *pangenomics* has emerged as a methodology to mitigate these shortcomings. The core idea is to replace reference genomes with more general reference structures that incorporate genomic variation across individuals. Including alternate alleles in the reference structure eliminates the privileged role of the reference allele, which is what leads to reference bias. Various approaches for doing so have been proposed, the majority of which have been based on *sequence graphs* (Figure 1.1). These graphs consist of vertices labeled with DNA sequences, which are connected by edges that indicate adjacency within a single DNA molecule. This structure allows sequences to split and rejoin around sites of variation. A full genome sequence can then be formed by concatenating sequences along a walk through a graph. In this way, the graph encodes a space of sequences rather than a single genome sequence.

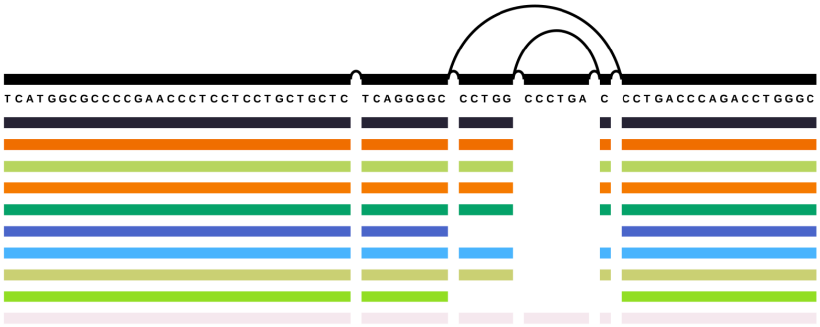


Figure 1.1: An example of a sequence graph. Colored bars indicate the walk taken by several haplotypes through the graph. The sequence of each haplotype can be reconstructed by concatenating the labels of the DNA sequences along the walk. Figure adapted from [70].

My thesis research has centered on developing methods for graph-based pangenomic analysis. Taken as a whole, the pangenomics project is rather ambitious. Reference genomes occupy a central role in genomics. Changing this core underlying formalism requires virtually every aspect of genomics analysis to be reconsidered, redesigned, and reimplemented. For this reason, it has been very satisfying to see field of pangenomics evolve over the course of my graduate education from a largely theoretical exercise to a mainstream methodology with growing practical significance, and I am humbled to have played a part in this maturation.

This dissertation summarizes what I consider to be my most significant contributions to pangenomics. Most of the chapters correspond to published works for which I share authorship. All of the chapters are intended to be standalone documents, and as such they can be read independently from each other. Chapter 2 provides a general overview of the field of pangenomics as well as some related fields. Chapter 3 describes methods I developed to apply pangenomic analysis to transcriptomics. Chapters 4, 5, and 6 describe contributions I made to the ecosystem of pangenomics software. Chapters 7 and 8 describe some of my research on applying graph theory to problems in pangenomics. Finally, Chapter 9 concludes with a discussion of the current state of pangenomics and its future outlook.

Chapter 2

Background

Pangenomics practitioners are exceptional borrowers. The field has thrived off of repurposing and recombining results from a range of adjacent fields. As a result, a background chapter for pangenomics tends to read like a primer on several distinct disciplines. In order to restrain the cacophony, I restrict my attention in this chapter to human genomics. This comports with my thesis research, which has similarly focused on humans.

Fittingly, parts of this chapter are also borrowed. The borrowed content comes from two review papers that I co-authored. The first, "Genome graphs and the evolution of genome inference", appeared in *Genome Research*. I was one of the primary co-authors along with Adam M. Novak and Benedict Paten [155]. The second, "Pangenome graphs", appeared in *Annual Review of Genomics and Human Genetics*. I co-authored this review primarily with Erik Garrison, along with smaller contributions from many co-authors [63].

2.1 Sequencing technology

The problem of sequencing DNA was first solved by Frederick Sanger in 1977 by using low-frequency incorporations of a modified, chain-terminating nucleotide into synthesizing DNA molecules [179]. The resulting shortened products could be separated and distinguished using electrophoresis and radiolabeling. Subsequent improvements to *Sanger sequencing* altered the method of separation and detection while preserving the underlying mechanism of chain-termination. The radiolabeling was replaced by fluorescent chain terminators, and gel electrophoresis was replaced by capillary electrophoresis [193]. The resulting technology produced highly accurate reads up to about 1000 bp in length.

Sanger sequencing remains useful and cost-effective for small-scale, targeted sequencing, but its low throughput is poorly matched to the scale of most problems in genomics. In most common genomics analyses, Sanger sequencing has now been largely supplanted by so-called *next-generation sequencing* (NGS) [136]. The term NGS initially encompassed several similar technologies, but today the market is dominated by Illumina's platform. This platform's chemistry generates clusters of identical copies of a DNA molecule using a variant of polymerase chain reaction (PCR). The clusters are then observed during a final round of synthesis in which the DNA molecules are reversibly terminated with a fluorophore-labeled nucleotide, one base at a time. This allows each subsequent nucleotide to be detected optically by its fluorescence. In addition, this process is massively parallel, yielding millions of 50-300 bp sequencing reads from a single sample.

Finally, two newer *third-generation sequencing* technologies achieve both high through-

put and long read length by sensing individual molecules rather than PCR products. The first from Pacific Biosciences (PacBio) uses sensitive optics to detect fluorescently-labeled nucleotides as they are incorporated into a DNA strand by an immobilized DNA polymerase [61]. The other from Oxford Nanopore Technologies (ONT) detects current fluctuations through a protein nanopore as a DNA molecule is pulled through it electrophoretically [138]. In both cases, the cost of the longer read length is much higher error rates, although it is possible to improve the accuracy of PacBio sequencing at the expense of some read length using a circular sequencing template [214]. Accordingly, NGS and third-generation technologies currently coexist on the sequencing landscape, each with their own niche of applications.

2.2 The human reference genome

The sequencing of a human genome was a landmark achievement. Over the span of decades, it was painstakingly sequenced using a combination of bacterial artificial chromosomes (BACs) and capillary Sanger sequencing [106]. In the following years, the genome assembly steadily improved [35,90] to the point that the Genome Reference Consortium's human genome assembly, GRCh38 [181], was arguably the best assembled mammalian genome in existence at the time of its release, with just 875 remaining assembly gaps and fewer than 160 million unspecified 'N' nucleotides (as of GRCh38.p8). In the near future, it is likely that this will be replaced by a reference that is essentially fully complete. The T2T Consortium recently announced a gapless telomore-to-telomere human genome assembly using a haploid human cell line [149].

Perhaps one reason the reference genome has been so effective as an organizing system is that the average human genome is remarkably similar to it. From short-read-based assays, it is estimated that the average diploid human has between 4.1 and 5 million small mutations (either single nucleotide variants (SNVs), multi-nucleotide variants (MNVs), or short indels), which is only around one variant every 1450 to 1200 bases of haploid sequence [39]. Such an average human would also have about 20 million bases—about 0.3% of the genome—affected by around 2,100-2,500 larger structural variants [39]. It should be noted that both these estimates are likely somewhat conservative as some regions of the genome are not accurately surveyed by the short read technology used. Indeed, long read sequencing demonstrates an excess of large variation not found by earlier short read technology [30, 183].

2.3 Read mapping

The first step in reference-based resequencing experiments is usually *read mapping*: the process of identifying the element of the reference genome that corresponds to each sequencing read. Generally speaking, a read's element-of-origin cannot be observed directly. Instead, read mapping tools use sequence similarity as a proxy. Mismatches between reads and the reference arise for two reasons: sequencing errors and genomic variation. Between humans' low rate of polymorphism and modern sequencing technologies' low error rates, the use of similarity as a proxy for identity usually works well.

Classical sequence alignment algorithms like Needleman-Wunsch [146] and Smith-Waterman-Gotoh [76,194] can identify similar DNA sequences. Moreover, the statistical theory

surrounding these algorithms is rich and well-developed [98, 196]. Unfortunately, these algorithms are unsuited to the scale of the read mapping problem. The run time of the classic algorithms scales with the product of the lengths of the sequences being compared. With a reference genome of $\sim$3 billion bp and sequencing data sets of $>$50 billion bp, they are clearly intractable.

Practical read mapping tools are invariably based on the *seed and extend* paradigm. This paradigm is built on two observations. First, the reference genome is static and repeatedly reused while mapping a set of reads. This makes it amenable to preprocessing techniques that accelerate the computation. Second, correct mappings are expected to be highly similar to the reference. Accordingly, the seed and extend paradigm begins by creating indexes of the reference genome that make it efficient to query exact matches. Then, each read can be used to query exact-match *seeds*, which are clustered and subsequently *extended* into full alignments using the classical algorithms. In the process, most of the reference genome can be filtered away and ignored.

The indexes used by seed-and-extend mappers can be divided into two main categories: *k-mer tables* and variants of the *suffix tree*. k-mer tables are usually implemented as hash tables that record the locations of k-mer sequences (subsequences of a fixed length k) in the reference [119, 123]. Various optimizations are used to restrain the hash tables' memory use and accelerate queries [59, 175]. The suffix tree is a classical data structure that encodes every suffix of a string as a walk from the root of a directed tree. If any such walks have matching prefixes, they are merged together. Suffix trees can locate matches of any length in linear time. They also require linear time and memory to construct [207], but the constant factor on

the memory usage can be prohibitive with large genomes [105]. *Suffix arrays* provide a more memory-efficient alternative that encodes the topology of the tree implicitly using only an array of integers [130]. The *FM-index* provides further space savings by allowing the reference to be stored in compressed form while still supporting suffix array queries [65]. This index has been widely used in read mapping software [107, 118].

2.4 Genome inference

Genome inference is the problem of determining an individual's genome sequence from DNA sequencing data. This process depends heavily on the characteristics of the sequencing technology being used. The most successful successful genome inference methods marry knowledge about the sequencing technology to computational and statistical techniques. These computational methods can be coarsely divided into two major categories: *de novo assembly* and *variant calling*.

2.4.1 De novo assembly

De novo genome assembly is the computational task of determining a sample's genome sequence directly from sequencing data without using a reference genome. For instance, the original human reference genome was produced by genome assembly techniques. In general, accurate genome assembly is considered a challenging problem. The core reason is that sequencing reads are much shorter than full chromosomes. Thus, to produce a full genome sequence, it is necessary to identify reads that originate from overlapping segments of the genome

so that they can be combined into longer sequences.

All modern assembly methods use sequence graphs to model uncertainty involved in process of identifying overlaps. The primary indication that two reads originate from overlapping segments of the genome is that their sequences match along an overlap. However, not all matches indicate that the reads originated from overlapping segments of the genome. Matches can also occur due to sequencing errors and repeated sequences in the genome. Thus, the matches between reads indicate a space of potential overlaps, and this space can be compactly summarized as a graph.

Two sequence graph formalisms are in common use for genome assembly: *overlap graphs* and *de Bruijn graphs*. In overlap graphs, each vertex corresponds to sequencing read, and edges are added whenever there is match between reads that is deemed statistically significant. These graphs are closely associated with the *overlap-layout-consensus (OLC)* framework for producing an assembly [142]. OLC formulates the assembly problem as producing a linear layout of the reads that respects the overlaps and produces consistent coverage across the layout. The read sequences can then be combined into a consensus sequence. *String graphs* are a closely-related variant that can be formulated as a modification of an overlap graph. First, nodes whose sequences are strictly contained in other nodes are removed, and then transitive edges are removed to produce a minimum equivalent graph (often incorrectly described as a transitive reduction) [143]. In this simplified graph, sequences that can be assembled unambiguously are easily identifiable as non-branching paths.

In de Bruijn graphs, each node's sequence corresponds to a *k-mer* found on a read. Edges are added whenever two k-mers are adjacent on some read [159]. The structure of a de

Bruijn graph contains strictly less information than an overlap graph. De Bruijn graphs only detect overlaps between reads when their k-mers match exactly, and they discard information about which k-mers co-occur on the same read, with the exception of directly adjacent pairs. However, the regular structure of de Bruijn graphs provides attractive benefits in computational efficiency.

2.4.2 Variant calling

In variant calling, genome inference is performed implicitly by describing the sample's genome as a collection of differences from a reference genome. This generally begins by mapping reads to the reference genome. The read sequences can then be compared to the reference sequence to identify differences. Compared to de novo assembly, the main advantage of variant calling is that it leverages the considerable information embodied in the reference. However, it also incurs reference bias for the same reason.

With NGS reads, small variants can be detected simply by counting the occurrences of a given reference mismatch among the mapped reads. This lends itself to count-based probabilistic models, which formed the basis of many successful early variant callers [73, 121]. The current state-of-the-art tools use the same underlying input, but replace explicit statistical modeling with deep neural networks [160].

The approach of counting instances of mismatches tends to fail with larger variants. Short reads containing large variants frequently cannot be successfully mapped to the reference (an example of severe reference bias). This fact motivates the distinction between *point variants* and *structural variants*, which are distinguished chiefly by their size. Conventionally, point

variants consist of variants that affect at most 50 bp, and structural variants are all larger variants. The boundary between the two is somewhat arbitrary, but it does mark the approximate size at which mapping biases make direct comparison of the read sequence untenable in NGS data.

Despite the difficulties in mapping, there are several strategies to call structural variants with NGS reads. Often these methods identify regions where a structural variant might exist by identifying aberrant features of read mappings. For example, copy number alterations (deletions or duplications) can be detected by abnormal coverage, which is actually an artifact of the reference having the wrong number of copies for the sample [1]. Other methods identify structural variants by finding regions where many reads align poorly. Sometimes it is possible to re-map the parts of the reads that are poorly aligned to identify deletions and inversions [110, 169]. Alternatively, the poorly-mapped reads can be used for *local assembly*: de novo assembly of a subset of reads based on their mapped location [161, 164, 174].

In contrast to NGS methods, long reads excel at detecting structural variants but struggle with point variants. The long read length often makes it possible to map reads containing structural variants. Thus, the basic strategy of counting mismatches in reads that is used to detect point variants in NGS can be used to detect structural variants [87, 182]. However, the high error rate makes it challenging to distinguish point variants from sequencing errors. A notable counterexample is PacBio HiFi reads, which are sufficiently high quality to detect small variants accurately [214]. It is also possible to obtain good point variant calling from ONT reads by leveraging haplotype phase [184].

2.5 Human population genomics projects

Many large-scale research efforts have undertaken the task of cataloguing the extant genomic variation across the human population using genome inference methods like the ones described above. The International HapMap Project was the first major example. It used largely array-based technologies in four geographically-diverse populations [42, 89], later expanded to 11 [43], with a total of 1,184 samples. The 1000 Genomes Project (1000GP) followed using low-coverage NGS in combination with other data types, which held the promise of discovering novel variants in addition to genotypes [37, 38]. The 1000GP eventually sampled 2,504 individuals across 26 populations [39].

No subsequent effort has matched the impact of the 1000GP, but further studies have expanded and refined the picture of global variation. This was done in part by using higher-coverage sequencing to ascertain rare variants [203] (including on the samples from the 1000GP [27]). Later projects also greatly expanded the number of populations included in global panels [18, 129, 152]. Finally, a growing number of regional and national sequencing projects have assayed variation intensively in specific geographic regions, including in Sweden [6], Mongolia [14], China [34], the Netherlands [21], Asia [41], Iceland [81], Africa [82, 186], Denmark [132], Japan [144], the United Kingdom [45], and others.

The vast majority of these projects have assayed variation using NGS data. Accordingly, they have made comparatively little progress at cataloguing structural variation relative to point variation [199]. However, studies are beginning to appear that use newer technologies. Several of these applied intensive sequencing efforts and de novo assembly methods to a

small number of sample genomes [12, 31, 57] (or even just one [183, 216]) to avoid reference bias. Only two population-scale long read sequencing projects have been conducted so far: one in Iceland with 3,622 samples [19] and another in China with 405 samples [216]. One further study used optical mapping, which does not produce sequence-resolved variant calls, to identify structural variation in 156 samples from across the 26 populations from the 1000GP [113].

Between these studies, the landscape of human population variation has been very well characterized. However, challenges remain that make it difficult to combine these data resources. Differences in assays and processing sometimes make the data incomparable. Furthermore, not all of the data are publicly available.

2.6 Pangenome graphs

As discussed in Chapter 1, most pangenomics methods substitute conventional reference genomes with sequence graph references. Full haplotype sequences can be formed by concatenating the DNA labels of the nodes along a walk through the graph. The sequence graphs used in pangenomics are very similar to those used in genome assembly. The most significant difference is that, unlike assembly graphs, the adjacencies in pangenome graphs are *blunt*, meaning that edges indicate direct adjacency between sequences with no overlap.

2.6.1 Constructing pangenome graphs

Most pangenome graphs are derived from either haplotype assemblies or population variant call data sets. With assemblies, the major challenge is identifying which regions of the

assemblies are homologous so that they can be merged in the final graph. Tools and techniques for interspecific whole genome alignment have be repurposed for this task [11,120,139]. Variant call data sets are generally more straightforward since the reference genome serves as a proxy for homology. Non-reference alleles can simply be added as diverging paths from the reference sequence [74, 167]. Counterbalancing the simplicity of this process, variant call-derived graphs carry greater risk of reference bias, and they may not be able to represent complex variation.

2.6.2 Alignment to sequence graphs

Classic algorithms like Smith-Waterman-Gotoh [194] do not directly apply to sequence graphs. However, the recurrence relations that drive their scoring and traceback routines can be extended to allow the alignment of sequences to acyclic sequence graphs, as popularized in partial order alignment (POA) [111]. Further generalizations support the alignment of sequences graphs to sequence graphs [78] and sequences to cyclic graphs [8, 141, 145]. It is notable that many of these findings have been independently rediscovered or refined by contemporary researchers [9, 94, 172]. Some earlier algorithms require restricted scoring functions to achieve efficiency [172], but recent contributions have used less restricted functions that produce more biologically meaningful alignments [94].

The graph alignment algorithms used in practice have become faster over time. POA had equivalent asymptotic run time to linear alignment but required acyclic graphs [111]. Later optimizations simply ran slower on general graphs [100]. Algorithms are now known with equivalent asymptotic run time even on general graphs [94]. In addition, researchers have developed modified algorithms that run quickly in the practical context of real-world computer

architectures [91, 93, 171].

2.6.3 Mapping to sequence graphs

One of the most significant drivers of recent progress in pangenomics has been the development of efficient mapping tools for pangenome graphs. Although these mapping tools all target sequence graphs, there are significant differences in the types of graphs that they handle. Several tools apply only to acyclic variation graphs formed by adding variants to a linear reference. Examples include GENOMEMAPPER [180], Seven Bridges' GRAPH GENOME ALIGNER [167], HISAT2 [103], and V-MAP [208]. In contrast, VG [74] and GRAPHALIGNER [173] appear to be the only tools with open ambitions of mapping to arbitrary variation graphs, including complex local and global topologies.

The majority of these tools emphasize mapping NGS data. GRAPHALIGNER and V-MAP are the only graph mapping tools designed long read sequencing data. While V-MAP also supports NGS reads, GRAPHALIGNER's seeding strategy limits it to long reads.

For indexing, most graph mapping tools have opted for some variation of a k-mer table. GRAPHALIGNER, GENOMEMAPPER, Seven Bridges' mapper, and V-MAP all use this strategy. The remaining mappers use succinct text indexes analogous to the FM-index. VG-MAP uses the GCSA2 [189] and a longest-common-prefix array, which enable very specific queries at the expense of high memory utilization. HISAT2 uses a modified GCSA [192] that also encodes the graph structure itself. This helps give HISAT2 an impressively low memory footprint at the expense of a somewhat more limited set of queries.

Most graph mappers also employ graph-based alignment algorithms. The exceptions

are GENOMEMAPPER, which aligns to all paths out from a seed, and HISAT2. The HISAT2 alignment algorithm relies on a complex set of heuristics that depend heavily on its exact match index. VG and V-MAP both employ some version of partial order alignment. Seven Bridges first searches for a near-exact match using an exponential depth-first search, applying partial order alignment if this fails. GRAPHALIGNER is the only mapper to incorporate the most recent research into cyclic graph alignment algorithms.

2.7 Transcriptomics

Transcriptome profiling with RNA-seq is one of the most common uses of NGS technology [140]. RNA-seq protocols first isolate RNA molecules and then reverse-transcribe them into cDNA molecules, which can be sequenced. Whereas genomic DNA sequencing protocols sample reads more-or-less uniformly across the genome, RNA-seq samples a genomic element proportionately to its transcript expression, which varies greatly from gene to gene. Thus, RNA-seq is a much more quantitative assay. This introduces a number of bioinformatic challenges that are tackled by different tools.

2.7.1 Splicing-aware read mapping

In eukaryotes, intronic sequences are spliced out of mRNA before it is translated into protein. Therefore, the correct mapping for a corresponding RNA-seq read should align to the elements on both sides of the splice junction. Classic sequence alignment algorithms are not well-suited to discover this kind of alignment. Their optimization functions would score

the splicing event as a long deletion, which would be strongly penalized. Instead, specialized mapping tools are required for RNA-seq data [54, 102, 104].

Several sources of information can be used to restrain the search space of alignments to make splicing-aware mapping practical. First, splice junctions occur only at specific, well-defined sequence motifs [25]. Second, intron lengths tend to be within a certain range of values [77]. Third, existing gene model annotations identify known splice junctions [54], and some read mapping tools can aggregate information across reads to identify unannotated splice junctions as well [104]. Most importantly, splice junctions are only plausible if there is a high-scoring alignment of the read sequence bridging them.

2.7.2 Expression quantification

The number of RNA-seq reads assigned to a transcript is a quantitative measurement of its expression. In order to make this measurement comparable across transcripts, it is also necessary to normalize the read counts by the length of the transcript [157, 211]. Estimating these normalized expression values can be formulated as mixture model and fit using the expectation-maximization (EM) algorithm [115]. Moreover, the EM algorithm also gracefully handles read mapping uncertainty. Some widely-used methods skip read mapping altogether and integrate an alignment-free assignment of reads to transcripts into the EM algorithm [23, 157]. This dramatically reduces the pipeline's computational requirements at the expense of some information loss.

2.7.3 Differential expression

The goal of many RNA-seq experiments is to identify differences in expression across different conditions. These conditions can be experimental perturbations or uncontrolled biological variables (such as tissues). Expression quantification is typically insufficient to identify differential expression on its own. Instead, point estimates of expression must be analyzed in statistical models that take into account technical and biological variability. A variety of statistical methodologies have been applied to this problem [109, 127, 176, 202].

Part II

Pangenomic analysis of the transcriptome

Chapter 3

A pipeline for pantranscriptome analysis

3.1 Preamble

What follows is the majority of the text of my preprint, ”Haplotype-aware pantranscriptome analyses using spliced pangenome graphs”, for which I share first-authorship with Jonas Sibbesen [187]. The paper details three components of a pantranscriptomic pipeline. The first component builds spliced pangenome graphs, the second component aligns RNA-seq data to these graphs, and the final component infer haplotype-specific isoform analysis from the mapping results. My primary contributions were to design and implement the mapping algorithm. In addition, I performed the genomic imprinting experiments, implemented the read simulation model, and contributed extensively to the writing and editing. The software development featured minor contributions from our co-authors Adam M. Novak, Xian Chang, Jouni Sirén, and Erik Garrison.

3.2 Introduction

Transcriptome profiling by RNA-seq has matured into a standard and essential tool for investigating cellular state. Bioinformatics workflows for processing RNA-seq data vary, but they generally begin by comparing sequencing reads to the sequence of a reference genome or reference transcriptome [23,54,114,157]. This is an expedient method that makes it practical to analyze the large volume of data produced by modern high-throughput sequencing.

Reference-based methods also have costs. When a sample's genome differs from the reference, bioinformatics tools must account for the resulting mismatches between the sequencing data and the reference. This results in reduced ability to correctly identify reads with their transcript-of-origin, with larger genomic variation leading to a greater reduction in accuracy. This problem is known as reference bias [198].

Computational pangenomics has emerged as a powerful methodology for mitigating reference bias. Pangenomics approaches lean heavily on abundant, publicly-available data about common genomic variation for certain species (notably including humans). These methods incorporate population variation into the reference itself, usually in the form of a sequence graph [40, 63]. Mapping tools for pangenomic references have demonstrated reduced reference bias when mapping DNA reads [33, 74, 167]. This in turn facilitates downstream tasks that are frustrated by mapping biases, such as structural variant calling [86, 188].

The sequence graph formalism used in pangenomics has an additional attractive feature for RNA-seq data: it can represent splice junctions with little modification. Without this benefit, RNA-seq mappers for conventional references must make use of sometimes elaborate

algorithmic heuristics to align over known splice junctions [54, 215]. Alternatively, they can map to only known isoforms, but this technique has difficulty estimating mapping uncertainty due to the re-use of exons across isoforms [107]. There is also evidence that population information can reduce reference bias problems that are particular to RNA-seq data. Accounting for population variation at splice-site motifs has been shown to aid in identifying novel splice sites [197].

The current methodological landscape in pangenomics is ripe to be extended to pantranscriptomics: using populations of reference transcriptomes to inform transcriptomic analyses. There is some precedent in a few existing transcriptomic methods that have used sequence graphs. AERON [170] uses splicing graphs and GRAPHALIGNER [173] to identify gene fusions. ASGAL [53] uses splicing graphs to identify novel splicing events. Finally, the pangenomic aligner HISAT2 [103] has its origins in the RNA-seq aligner HISAT [102], and it retains many of HISAT's features for RNA-seq data. The performance of HISAT2 for pantranscriptomic mapping has not yet been characterized in published literature.

One transcriptomic analysis that is particularly prone to reference bias is allele-specific expression (ASE). ASE seeks to identify differences in gene expression between the two copies of a gene in a diploid organism. These differences are indicative of various biological processes, including cis-acting transcriptional regulation, nonsense-mediated decay, and genomic imprinting [28, 220]. The differences are identified by measuring the ratio between RNA-seq reads containing each allele of a heterozygous variant. However, the reads containing the non-reference allele are systematically less mappable because of reference bias, which can lead to both degraded and illusory signals of ASE [52]. Several approaches have been developed to

deal with reference bias for ASE detection. Some methods filter out biased sites [209]. Others can mitigate bias at the read mapping stage, but require variant calls, often with phasing, for the individual being analyzed [137, 166, 178]. The variant information is either incorporated into the mapping algorithm to reduce reference bias or used to create a sample-specific diploid reference to map against. PHASER phases called genotypes using read-backed and population-based phasing to produce estimates of haplotype-specific gene expression [29].

Pantranscriptomic approaches using existing haplotype panels for inferring haplotype-specific expressions have also been developed. ALTHAPALIGNR maps reads to the linear reference genome and seven alternative HLA haplotypes to infer haplotype-specific transcript expression in the HLA region [112]. HLAPERS first aligns reads against all known HLA haplotypes to estimate the most likely haplotypes and then infers haplotype-specific gene expression [2]. However, both of these pantransciptomic approaches are limited to smaller genomic regions.

In this work, we present a novel bioinformatics toolchain for whole genome pantranscriptomic analysis, which consists of additions to the vg toolkit and a new standalone tool, RPVG. First, the VG RNA tool can combine genomic variation data and transcript annotations to construct a spliced pangenome graph. Next, VG MPMAP can align RNA-seq reads to these graphs with high accuracy. Finally, RPVG can use the alignments produced by VG MPMAP to quantify haplotype-specific transcript expression. Moreover, the information about population variation that is embedded in the pantranscriptome reference makes it possible to do so without first characterizing the sample genome, and without restricting focus to SNVs.

3.3 Results

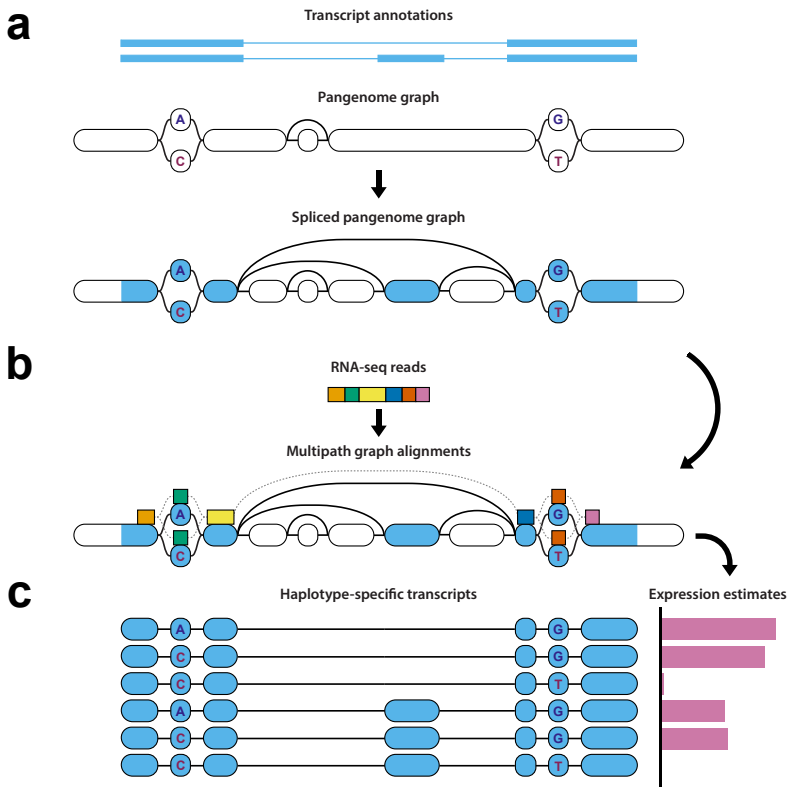


Figure 3.1: Diagram of haplotype-aware transcriptome analysis pipeline
The three major steps in the pipeline. **a** VG RNA adds splice junctions derived from a transcript annotation to a pangenome graph to create a spliced pangenome graph. It simultaneously creates a pantranscriptome composed of a set of haplotype-specific transcripts (HSTs) using a panel of known haplotypes (not shown). **b** VG MPMAP aligns RNA-seq reads to subgraphs of the spliced pangenome graph represented as a multipath alignment. **c** RPVG uses the alignments from MPMAP to estimate the expression of the HSTs in the pantranscriptome.

3.3.1 Haplotype-aware transcriptome analysis pipeline

In short, our pipeline works as follows (see Methods for a more detailed description). First, we construct a spliced pangenome graph using VG RNA, a method developed as

part of the vg toolkit [74]. VG RNA adds splice junctions from a transcript annotation into a pangenome graph as edges and then labels the paths in the graph that correspond to transcripts (Figure 3.1a). Simultaneously, VG RNA constructs a set of haplotype-specific transcripts (HSTs) from the transcript annotation and a set of known haplotypes by projecting the transcript paths onto each haplotype. VG RNA uses the Graph Burrows-Wheeler Transform (GBWT) to efficiently store the HST paths allowing the pipeline to scale to a pantranscriptome with millions of transcript paths [190]. Next, RNA-seq reads are mapped to the spliced pangenome graph using VG MPMAP, a new splice-aware graph mapper in the vg toolkit that can align across both annotated and unannotated splice junctions (Figure 3.1b). VG MPMAP produces multipath alignments that capture the local uncertainty of an alignment to different paths in the graph (Supplementary Figure A.1). Lastly, the expression of the HSTs are inferred from the multipath alignments using RPVG (Figure 3.1c). RPVG uses a nested inference scheme that first samples the most probable underlying haplotype combinations (e.g. diplotypes) and then infers the HST expression using expectation maximization conditioned on the sampled haplotypes.

3.3.2 RNA-seq mapping benchmark

We compared VG MPMAP against three other mappers: STAR [54], HISAT2 [103] and VG MAP [74]. STAR and HISAT2 can both use splicing information to guide mapping, but of the two only HISAT2 is able to also utilize genomic variants. VG MAP is not a splice-aware mapper, but it is still able to map to spliced pangenome graphs, which contain both splicing and genomic variation edges.

We used two different references for the comparison: the standard reference genome

with added splice junctions (spliced reference) and a spliced pangenome graph containing both splice junctions and variants (spliced pangenome graph). For STAR only the spliced reference was used.

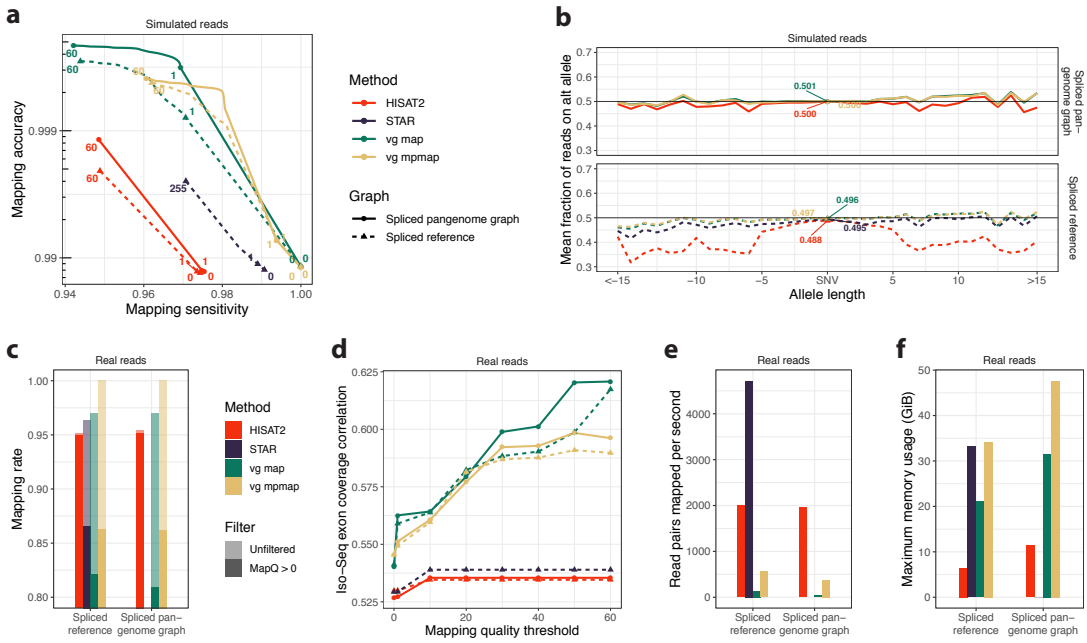


Figure 3.2: Mapping benchmark using RNA-seq data from NA12878
RNA-seq mapping results comparing VG MPMAP and three other methods using simulated and real Illumina data ("vg sim (ENC, uniform)" and "ENCSR000AED" in Supplementary Table A.4 and A.3, respectively). Solid and dashed lines show the results using a spliced pangenome graph and spliced reference, respectively. **a** Mapping accuracy and sensitivity for different mapping quality thresholds (colored numbers) using simulated data. An alignment is considered correct if it covers 90% of the true reference sequence alignment. **b** Mean fraction of mapped reads supporting the non-reference allele for variants of different lengths in simulated data. Negative lengths correspond to deletions and positive to insertions. The colored numbers are the mean fraction for SNVs. **c** Mapping rate using real data. The shaded bars show the mapping rate for all alignments and the solid bars for only alignments with a mapping quality above 0. **d** Pearson correlation between Illumina and Iso-Seq exon coverage using real data as a function of mapping quality threshold. Exons are defined by the Iso-Seq alignments. **e** Number of read pairs mapped per second per thread using real data. The mapping times were estimated using 16 threads on a AWS m5.4xlarge instance. **f** Maximum memory usage for mapping in gigabytes using real data.

3.3.2.1 Simulated sequencing data

Paired-end reads were simulated from HSTs derived from the GENCODE transcript annotation set [66] and the NA12878 haplotypes from the 1000 Genomes Project (1000GP) [39]. VG SIM was used to simulate the reads using reads from the ENCODE project (ENCSR000AED replicate 1) to parameterize the noise model [44, 49]. Fragment length distribution parameters used in the simulation were estimated from the same reads using RSEM [114]. The CEU population was excluded from the spliced pangenome graph as NA12878 is from that population, and we wanted to estimate performance for a new individual, who may not be as closely-related to the 1000GP populations. We simulated the HSTs with uniform expression rather than trying to match a previous expression profile, which could bias expression towards easily-mappable transcripts.

Using the set of simulated reads we first compared the overall mapping performance of each method. Figure 3.2a shows the mapping sensitivity and accuracy (log-scale) for different mapping quality thresholds. An alignment was considered correct if it covered 90% of the true reference sequence alignment. The graph alignments from VG MAP and VG MPMAP were projected to the reference sequence for this comparison. As can be seen in Figure 3.2a, VG MPMAP achieves both a high sensitivity and accuracy, while the other methods either had a lower accuracy or sensitivity. The results also show that the spliced pangenome graph generally improves mapping performance.

To evaluate the method's ability to align over unannotated splice junctions, we re-

peated the experiment on a spliced pangenome graph (spliced reference for STAR) created from an annotation set with 20% of the transcripts missing (Supplementary Figure A.2). This number was based on recent estimates of the fraction of novel transcripts in a sample using long reads [217]. As expected, the performance of VG MAP decreases dramatically since it can only align over splice junctions represented in the graph. VG MPMAP's performance decreased markedly more on the downsampled annotation compared to STAR and HISAT2 using the 90% threshold (Supplementary Figure A.2a). This is likely due to its more conservative approach to finding novel splice-junctions that require a high-scoring alignment in order for a new junction to be statistically significant. Indeed, when using a threshold of 70%, the mapping accuracy VG MPMAP increases to a value higher than STAR and HISAT2 even when they use the complete transcript set (Supplementary Figure A.2b).

Next, we looked at whether using a variant-aware approach reduces reference bias. Figure 3.2b shows the mean fraction of reads mapped to the alternative allele for different allele lengths. Negative values correspond to deletions and positive values to insertions. When using the spliced reference genome, all methods exhibit a bias towards the reference allele, with VG MAP and MPMAP showing less bias than the other methods. Using the spliced pangenome graph results in substantially reduced bias for all methods.

The mapping results were corroborated by an alternate correctness criterion based on aligning within 100 bases of the correct position on the paths in the graph (Supplementary Figure A.3).

The set of simulated reads used for the mapping evaluation was not used to optimize the de-

velopment and parameters of VG MAP and MPMAP. Supplementary Figure A.4a shows the results on the dataset that was used for optimization, which used RNA-seq data from Tilgner et al. [204] to parameterize the read simulation. The sensitivity and accuracy estimates for MPMAP are quite similar between the two datasets indicating that MPMAP is not overfit to the training data. The performance of all the other methods was generally worse on the training data.

3.3.2.2 Real sequencing data

We used the same read set from the ENCODE project that was used to parametrize the simulations to benchmark the methods on real data. We first looked at the fraction of aligned reads for each method (Figure 3.2c). VG MPMAP was able to map more reads overall than any of the other methods. HISAT2 achieved a higher mapping rate for mappings with mapping quality greater than 0, but seemingly at the cost of low specificity (Figure 3.2a). Using the spliced pangenome graph did not have a notable influence on the mapping rates.

Ground-truth alignments are not available for real data, so we use a proxy based on Pacific Biosciences (PacBio) Iso-Seq read mappings instead. Specifically, we compare to Iso-Seq read alignments generated by the ENCODE project (ENCSR706ANY) from the same cell line as the Illumina reads. Since the cell line is the same, we expect the transcript expression to be similar. Moreover, long reads can generally be mapped more confidently than short reads. Thus, higher correlation in coverage between short read mappings and the Iso-Seq mappings should be indicative of more accurate short read mappings in the aggregate. Figure 3.2d shows the estimated Pearson correlation in the coverage of each exon as a function of mapping quality threshold. As can be seen, both VG MAP and MPMAP achieves higher correlation than STAR

and HISAT2, with the spliced pangenome graph resulting in even higher correlation for both. The graph alignments from VG MAP and MPMAP were projected to the reference genome for this analysis.

Finally, we compared the methods' mapping speed and memory usage. Figure 3.2e shows the number of read pairs mapped per second per thread. Conversion from SAM to BAM was included in the HISAT2 time estimate to be more comparable to the output type of the other methods. VG MPMAP's increase in accuracy does not come for free; it is between 3.6 and 5.2 times slower than HISAT2, depending on the graph. However, it is 9.4 times faster than VG MAP on the spliced pangenome graph. VG MPMAP uses slightly more memory than STAR (Figure 3.2f).

We also compared the mapping performance of the different methods on the real RNA-seq data from Tilgner et al. [204] and CHM13 RNA-seq data from the T2T consortium (Supplementary Figure A.4b,c and A.5). Similar overall tendencies are observed using these datasets. It is important to mention that the CHM13 data was used during the development of VG MPMAP, and the other set was used to optimize the parameters of VG MAP and VG MPMAP.

3.3.3 Haplotype-specific transcript quantification

We compared RPVG to three other transcript quantification methods: KALLISTO [23], SALMON [157] and RSEM [114]. We stress that none of these methods were developed to work on pantranscriptomes with millions of HSTs. However, they serve as a point of reference for what accuracy is achievable without new methods development. The simulated data was gener-

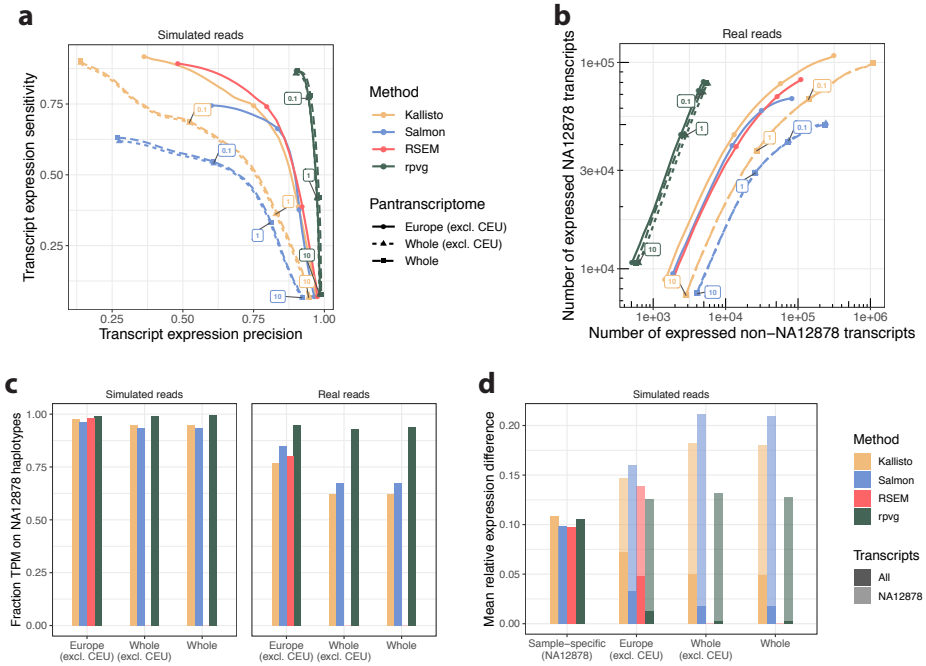


Figure 3.3: Haplotype-specific transcript quantification benchmark using RNA-seq data from NA12878

Haplotype-specific transcript (HST) quantification results comparing RPVG against three other methods using simulated and real Illumina data ("vg sim (ENC, RSEM)" and "ENCSR000AED" in Supplementary Table A.4 and A.3, respectively). Solid lines with circles are results using a pantranscriptome generated from 1000 Genomes Project (1000GP) European haplotypes excluding the CEU population. Dashed lines with triangles and squares are results using a pantranscriptome generated from all 1000GP haplotypes without and with the CEU population, respectively. **a** Sensitivity and precision of whether a transcript is correctly assigned nonzero expression for different expression value thresholds (colored numbers for "Whole (excl. CEU)" pantranscriptome) using simulated data. Expression is measured in transcripts per million (TPM). **b** Number of expressed transcripts from NA12878 haplotypes shown against the number from non-NA12878 haplotypes for different expression value thresholds (colored numbers) using real data. **c** Fraction of transcript expression (in TPM) assigned to NA12878 haplotypes for different pantranscriptomes using simulated (left) and real (right) data. **d** Mean absolute relative difference (MARD) between simulated and estimated expression (in TPM) for different pantranscriptomes using simulated data. MARD was calculated using either all HSTs in the pantranscriptome (solid bars) or using only the NA12878 HSTs (shaded bars). "Sample-specific (NA12878)" is a personal transcriptome generated from 1000GP NA12878 haplotypes.

ated by VG SIM, largely as described for the mapping benchmark. The only difference was that, rather than simulating transcripts with uniform expression, we simulated according to an expression profile that was estimated by RSEM using the same ENCODE reads. Three different pantranscriptomes were generated for the evaluation using different sets of 1000GP haplotypes (Supplementary Table A.2): 1) all European haplotypes excluding the CEU population (Europe (excl. CEU)) 2) all haplotypes excluding the CEU population (Whole (excl. CEU)) and 3) all haplotypes (Whole). The CEU population was excluded for the same reason as in the mapping benchmark: because NA12878 is part of this population, and we wanted to evaluate the realistic setting in which a sample is not as well represented by the haplotype panel. In addition, we created a sample-specific transcriptome consisting of NA12878 HSTs (Sample-specific (NA12878)). This transcriptome corresponds to the ideal case where a sample's haplotypes are known beforehand.

We first looked at the method's ability to accurately predict whether an HST was correctly expressed or not. Figure 3.3a shows the sensitivity and precision of whether a transcript is correctly expressed or not using simulated data. The results were stratified by different expression thresholds up to a value of 10 TPM (transcripts per million). Note that we were not able to run RSEM on the two largest pan-transcriptomes used in the figure. RPVG exhibits a mucher higher precision and sensitivity than the other tools for all pantranscriptomes. Over 97.4% of the HSTs with an expression value of at least 1 TPM are correctly predicted to be expressed by RPVG using the "Whole (excl. CEU)" pantranscriptome. Importantly, only a minor difference is observed between the pantranscriptomes without the CEU population (excl. CEU) and the

whole pantranscriptome (Whole), which contains NA12878. This could be explained by the fact that less than 2% of HSTs are on average unique to a specific sample when compared to all samples in other populations using the 1000GP data (Supplementary Figure A.6). This suggests that haplotype panels like the 1000GP are a good alternative when a sample's haplotypes are not available, although there are always limits to panel diversity, and some samples will be less well-represented by a 1000GP pantranscriptome.

We also evaluated the accuracy of the HST expression estimation using real sequencing data (Figure 3.3b). Since we do not know which transcripts are expressed in real data, we focus instead on the haplotype estimation. Sample NA12878's haplotypes are known to a reasonably high degree of certainty. Thus, we can indirectly measure accuracy by asking whether the HSTs that are estimated to be expressed are in fact from NA12878. Similar to the simulated data, Figure 3.3b shows that RPVG predicts markedly fewer HSTs from non-NA12878 haplotypes than both KALLISTO and SALMON. Using the "Whole (excl. CEU)" pantranscriptome, RPVG predicted 2,836 HSTs from non-NA12878 haplotypes to have an expression value of at least 1 TPM, while SALMON and KALLISTO predicted 25,790 and 26,779, respectively.

Next, we compared the fraction of transcript expression (in TPM) that was attributed to NA12878 haplotypes. This is shown in Figure 3.3c for both simulated (left bars) and real (right bars) data. We see that RPVG attributes more than 98.9% and 93.1% of the expression to NA12878 haplotypes when using simulated and real data, respectively. Furthermore, the figure shows that RPVG's prediction accuracy only decreases slightly when the size of the pantranscriptome increases from 2.5M HSTs in "Europe (excl. CEU)" to 11.6M in "Whole (excl. CEU)".

We compared how well the different methods could predict the correct expression value. Figure 3.3d shows the mean absolute relative difference (MARD) between the expression values of the simulated reads and the estimated values. The solid bars are MARD values when using all HSTs in the pantranscriptomes, and the shaded bars are when comparing the NA12878 HSTs only. Note that these bars are the same for the sample-specific set, which consists of only NA12878's HSTs. On the sample-specific set, RPVG performs comparably to the other methods. However, as the size of the pantranscriptome grows, the increase in MARD on the NA12878 transcript set is considerably less for RPVG relative to the other methods. When looking at the whole transcript set in each pantranscriptome (solid bars), RPVG has the lowest MARD. The much lower values compared to the NA12878 transcript set can be explained by the large number of unexpressed HSTs in the full pantranscriptomes.

We also compared the expression values using Spearman correlation (Supplementary Figure A.7). This metric supported overall similar conclusions, albeit with KALLISTO and RSEM performing comparably to RPVG when using the pantranscriptomes but restricting focus to NA12878's haplotypes. This suggests that KALLISTO and RSEM accurately rank these transcripts' expression but do not accurately estimate the absolute quantity.

To show the advantage of the multipath alignment format when inferring HST expression we repeated the evaluation using single-path alignments as input to RPVG (Supplementary Figure A.8). The single-path alignments were created by finding the best scoring path in each multipath alignment. For all pantranscriptomes and datasets, RPVG gave the best results using

the multipath alignments.

Similarly to the mapping benchmark, we also evaluated RPVG on RNA-seq data from the CHM13 cell line and NA12878 RNA-seq data from Tilgner et al. [204] (Supplementary Figure A.9 and A.10). Overall, similar conclusions can be drawn using these data. It is important to mention that both datasets were used to optimize the parameters of RPVG.

3.3.4 Assaying isoform-specific genomic imprinting

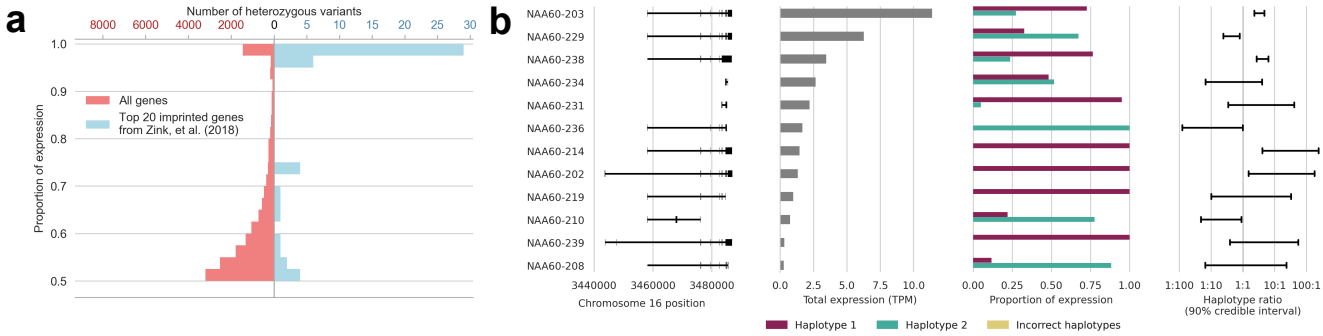


Figure 3.4: Exploratory demonstration of analyzing genomic imprinting using data from GM12878 lymphoblastoid cell line
Results of the VG MPMAP-RPVG pipeline on RNA-seq data from a lymphoblastoid cell line from the ENCODE Project, focusing on genes previously identified as imprinted in blood. **a** The proportion of expression attributed to the higher-expressed allele of heterozygous variants among the 20 most significantly imprinted genes from Zink's, et al. study [220] compared to all genes. The axes are scaled so that both histograms have the same area. **b** Isoform-level allele specific expression in NAA60, which was previously identified as imprinted but not as having isoform-dependent reversals in the polarity of imprinting [220]. Isoforms with expression less than 0.25 transcripts per million (TPM) are not shown.

To demonstrate the utility of the VG MPMAP-RPVG pipeline, we performed an exploratory analysis of genomic imprinting in a human sample. Genomic imprinting is the phenomenon in mammals in which some genes are expressed only from the copy inherited from a specific parent. That is, either the maternal or paternal copy is silenced, regardless of the ge-

nomic sequence of that haplotype. This is accomplished through mitotically heritable epigenetic modifications that are established during early development [206].

Several previous studies have studied imprinting genome-wide by quantifying ASE in RNA-seq data. These studies have demonstrated that imprinting varies substantially across tissues [13,16] and varies in intensity across genes, with many genes showing biased expression away from one parent-of-origin copy but not complete silencing [13, 92, 220]. In addition, a handful of genes have been identified in which the polarity of imprinting depends on the isoform: some isoforms of the same gene are biased toward the paternal copy and others toward the maternal copy [220].

The previous genome-wide studies have methodological limitations that diminish their ability to detect isoform-level imprinting. Some have aggregated ASE across all isoforms of the gene, which precludes isoform-level analysis a priori [13, 16, 92]. The largest study, by Zink, et al. [220], performed all tests of imprinting on individual SNVs. This method can sometimes detect isoform-level differences if the isoforms have some unshared exons. However, in shared exons, the ASE signal from the highest-expressed isoforms can drown out the signal of lower-expressed isoforms. Depending on the configuration of exons in the isoforms, this can make it very challenging to identify imprinting of opposite polarity within the same gene.

Figure 3.4 shows results from our exploratory demonstration of isoform-level imprinting analysis using VG MPMAP and RPVG. We ran the entire pipeline using RNA-seq data from a lymphoblastoid cell line derived from 1000 Genomes Project sample NA12889, which was sequenced as part of the Geuvadis project [108]. As a confirmatory analysis, we first looked for ASE in genes that have previously been identified as imprinted. In particular, we looked

at the 20 genes with the most significant p-values from Zink's, et al. study [220]. To mirror the methods used in this paper, we derived variant-specific ASE values by aggregating the expression across all HSTs that contain a given allele for a variant. Figure 3.4a shows that the VG MPMAP-RPVG pipeline detects ASE at heterozygous variants in these imprinted genes at a markedly higher rate than in background across all genes.

The VG MPMAP-RPVG pipeline is also capable of detecting isoform-dependent genomic imprinting. Figure 3.4b shows an illustrative example in the gene NAA60, which codes for an enzyme that acetylates the N-terminus of proteins in the Golgi apparatus. The isoforms show a complex pattern of imprinting polarity, which does not correlate strongly with exons or start sites in any particular genomic region. Given the large differences in expression of these isoforms, the SNV-based analysis would have had difficulty identifying imprinting in the more lowly-expressed isoforms, and indeed this gene was reported as imprinted (in fact, it is among the 20 most significantly genes referred to above) but not as having isoform-dependent imprinting [220]. It should be emphasized that this exploratory analysis of a single sample, while suggestive, is insufficient to conclusively demonstrate isoform-dependent imprinting in NAA60. Doing so would require further biological replicates and more rigorous controls for cis-regulation and cell line clonality [92].

3.4 Discussion

The pace of development in the field of eukaryotic pangenomics has surged in recent years. Improvements in sequencing technology have made it practical to characterize the

genomes of increasingly many samples. As a result, pangenomes made from tens to hundreds of reference-quality genome assemblies have been constructed for several agricultural organisms [48, 97, 126], and similar efforts are underway for humans by the Human Pangenome Reference Consortium and others [57]. Simultaneously, the bioinformatics tools to do pangenomic analyses have matured to the point of practicality for many applications [86, 131, 191]. Moving forward, we anticipate that pangenomic methods will continue to expand to inform increasingly many areas of genomics [79].

In this work, we have presented one step in this expansion: generalizing transcriptomics into pantranscriptomics. Our novel bioinformatics pipeline provides a full stack of tools for pantranscriptomic analysis. It can construct pantranscriptomes, map RNA-seq reads to these pantranscriptomes, and quantify transcription with haplotype-resolution. The construction takes advantage of efficient pangenome data structures, the mapping achieves a desirable balance of accuracy and speed, and the quantification can infer haplotype-specific transcript expression even when the sample's haplotypes are not known beforehand.

Some downstream applications are already apparent. For one, the pipeline can be used to study causes of haplotype-specific differential expression. We demonstrated its capabilities on one such example: genomic imprinting. The demonstration showed suggestive evidence of complex patterns of imprinting at the isoform level, which would have difficult or impossible to detect with previous genome-wide methodologies. The pipeline could be similarly used to study other sources of haplotype-specific expression, such as nonsense-mediated decay and cis-regulation.

Another application is characterizing genotypes and haplotypes in coding regions

from RNA-seq data. This could give access to the exact transcript sequences in a sample's transcriptome. In both of these applications, this pipeline increases the information that is available from RNA-seq data without paired genomic sequencing. This will enable low-cost study designs and deeper reanalyses of existing data.

Of course, our pipeline also has limitations. We have developed it to have good performance on pantranscriptomes constructed from phased variant calls. This is presently the most available data resource for constructing pangenomes. However, as increasingly many haplotype-resolved assemblies are produced, we predict that the emphasis in pangenomics will shift to pangenome graphs constructed from whole genome alignments. Constructing these graphs is currently an area of active research [96, 120]. Such graphs have more complicated topologies, often involving complex cyclic motifs. Experience leads us to believe that pantranscriptomic tools will require further methods development to use these data resources effectively.

Our pipeline is optimized for short-read RNA-seq data. The higher-error long-read RNA-seq technologies developed more recently require specifically-tailored algorithms for efficient analysis [119, 217]. Pantranscriptomic analyses of long-read RNA-seq data will likewise require further development, although the pipeline described here could serve as a platform for this development. Nevertheless, the cost-effectiveness of short-read sequencing virtually ensures that it will remain an important part of the sequencing landscape into the near future. Finally, our pipeline also relies on having a comprehensive pantranscriptome that contains many of the sample's haplotype-specific transcripts. The pantranscriptomes used in this study (based on the 1000 Genome Project) provided good results in the three samples analyzed,

but this performance may not extend to samples from other populations. Here—and throughout pangenomics—there is a compelling case to improve the completeness of data resources through more diverse sampling.

3.5 Acknowledgements

Research reported in this chapter was supported by the National Human Genome Research Institute of the National Institutes of Health under Award Numbers U01HG010961 and R01HG010485. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. The work of JAS was supported by the Carlsberg Foundation. We thank the ENCODE Consortium, the Thomas Gingeras Laboratory (Cold Spring Harbor Laboratory) and the Ali Mortazavi Laboratory (University of California Irvine) for generating and sharing the ENCODE data used in this study. We would also like to thank Megan Dennis (University of California Davis) for generating and providing access to the CHM13 RNA-seq data on behalf of the T2T consortium. Finally, we would like to thank Jean Monlong and Glenn Hickey for feedback on the manuscript, and everybody else in the vg team.

3.6 Methods

3.6.1 Sequencing data, transcript annotations and variation databases

GENCODE v29 (primary assembly) was used as a transcript annotation set [66]. All transcripts with either the mRNA_start_NF or mRNA_end_NF tag were removed in order to only

keep confirmed full-length transcripts. Furthermore, a transcript subset containing 80% of the GENCODE transcripts was created by randomly removing 34,490 of the 172,449 transcripts in the annotation. The fraction removed was based on recent estimates of the fraction of novel transcripts in a sample using long reads [217].

Genomic variants on GRCh38 from the 1000 Genomes Project (1000GP) were downloaded from EBI (`http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/GRCh38_positions/`) [39]. The variants were first normalized using BCFTOOLS [117] and four different sets containing variants from differently-sized collections of samples were created (Supplementary Table A.1). Two of these sets were constructed so as to not include variants unique to the CEU population. This was because we benchmarked the pipeline on NA12878, who is from this population, and we wanted our evaluations to cover one of the intended use-cases for the pipeline: sequencing a new sample from a population that is not represented in the reference haplotype panel. For all of the variant sets except the sample-specific set (where allele frequency was not relevant), the intronic and intergenic variants were further filtered using BCFTOOLS, keeping only variants with an alternative allele frequency of at least 0.002 or 0.001 depending on the set. This was done to decrease the complexity of the graph in regions where fewer reads are expected to map. The GRCh38 (primary assembly) reference genome used throughout the study was downloaded from Ensembl (`ftp://ftp.ensembl.org/pub/release-94/fasta/homo_sapiens/dna/`).

A list of all sequencing data used can be found in Supplementary Table A.3.

3.6.2 Spliced pangenome graph construction

We developed a method in the vg toolkit, VG RNA, for constructing spliced pangenome graphs from a transcript annotation and an existing pangenome graph. VG RNA begins by identifying the path in the graph that corresponds to each exon in the annotation. This process is facilitated by indexes in the vg toolkit that can efficiently query graph locations of positions on the linear reference. These exon paths can start or end internally in a node rather than only at boundaries between nodes, as with other paths in vg. Next, VG RNA divides nodes as necessary to expose exon boundaries as node boundaries and then adds edges (splice-junctions) to the graph connecting adjacent exons within each transcript. The transcript paths are then labeled in the resulting spliced pangenome graph. Lastly, the spliced pangenome graph's node ID space is compacted and reordered in topological order to make graph compression more efficient [62].

In addition to the spliced pangenome graphs, VG RNA was also used to construct exon-only splicing graphs. VG RNA creates these graphs by removing all nodes and edges from a spliced pangenome graph that were not covered by a transcript path. Different combinations of transcript annotations (full and an 80% random subset) and variant sets (Supplementary Table A.1) were used as input to create the graphs used in the mapping and expression inference evaluation.

3.6.3 Pantranscriptome construction

In addition to constructing spliced pangenome graphs, VG RNA can simultaneously generate pantranscriptomes consisting of haplotype-specific transcripts (HSTs) created from transcript and haplotype annotations. It creates pantranscriptomes by projecting the reference

transcript paths onto haplotypes paths that are either labeled in the graph or indexed using the Graph Burrows-Wheeler transform (GBWT) [190]. The GBWT is a succinct data structure for efficiently storing thousands of paths in a graph, such as haplotypes or transcripts. If nodes are split during the spliced pangenome graph construction (see above), VG RNA first updates the haplotypes in the input GBWT. Next, the flanking positions of the exon boundaries on the reference chromosome path are located in the graph. These positions are used as anchors for projecting exons between the reference and haplotype paths. Anchoring on the positions adjacent to exon boundaries allows for genomic variation at the distal ends of exons.

Depending on whether the haplotype paths are labeled in the graph or stored in a GBWT, the projection is performed differently. For haplotype paths that have been labeled in the graph, we first locate all paths that contain both anchor nodes for each exon in a transcript. Next, for each located exon anchor pair we then follow the haplotype path between the two anchors to create the projected haplotype-specific (HS) exon path. HST paths are then created by combining all HS exon paths that are projected to the same haplotype. Only complete transcripts where all exons are successfully projected are kept. A projection will fail if there is variation at the anchor position in the target haplotype. Finally, HST transcripts that are identical are collapsed, producing a set of unique HSTs for each reference transcript. Since the number of pre-collapsed HSTs can be as large as the number of haplotypes, the algorithm is not suitable for large haplotype sets. For these, the GBWT-based algorithm, described below, is a better choice.

A broadly similar approach is used when the haplotype paths are stored in a GBWT. However, it differs in how the projected exons paths are constructed and combined. To find

all possible haplotype paths between two exon anchors, we use a depth-first search (DFS). The search is initialized at the start anchor and traverses all possible paths in the graph starting from that anchor. Each explored exon path in the DFS (branch) is queried against the GBWT index and is terminated if it is not a subpath of any haplotypes in the index. Furthermore, a branch is also terminated if it is not possible to reach the end anchor node by any of the haplotypes consistent with the exon path. This is determined by examining whether any of the haplotypes containing the exon path also contain the end anchor node. The output from the search is a list of unique projected HS exon paths and the set of haplotypes consistent with each of them. The final HST paths are constructed one exon at a time by connecting HS exon paths that share at least one haplotype for each transcript. Because all the HS exon paths are unique this procedure will always result in a set of only unique HST paths and thus it is not necessary to collapse identical paths. This attribute makes the approach using the GBWT scale well with the number of haplotypes, as it can take advantage of the fact that haplotypes are often identical locally.

A list of all pantranscriptomes created for this study including the transcript annotations and variant sets used as input can be seen in Supplementary Table A.2. The HSTs were written both as nucleotide sequences in FASTA format and as paths to a GBWT. A bidirectional GBWT, where each path is stored in both directions, was also created. RPVG uses this index to decrease computation time when reads are not strand-specific. For each GBWT, a corresponding *r*-index was further constructed. This index, based on the original *r*-index by Gagie et al., significantly decreases the computation time it takes to query path IDs in the GBWT [67].

3.6.4 Read simulation model

Simulated reads were generated using VG SIM, a read simulator in the vg toolkit that is designed primarily for next-generation sequencing (NGS) reads. Its model consists of three components: a Markov model for base quality strings, a path frequency model, and a fragment length model (when sampling paired-end reads).

The model for base quality strings is fit to replicate the base quality strings in a user-provided FASTQ. A separate Markov transition distribution is fit for each base position in the read. The state of each Markov distribution consists of two components: the Phred base quality at that base and whether that base is an N. If a paired-end FASTQ is provided, VG SIM will fit a separate model for each read end. In addition, the first states of each read in the pair are modeled with a single joint distribution, which allows for some dependence between the quality of both reads in the pair. The probabilities of the Markov transitions and the initial states are estimated by their empirical frequency in the FASTQ.

VG SIM determines the base sequence of each read by following random walks through the pangenome graph. These walks may optionally be restricted to specific paths through the graph. Importantly for this study, the simulation can be restricted to the paths of transcripts in a spliced pangenome graph. The sampling frequency of a transcript path is proportional to the product of its length and its expression value measured in transcripts per million (TPM), as determined by a user-provided expression profile. Once the path has been chosen, the starting location of the read is selected uniformly at random along the transcript. The sequence of the walk is then extracted, and sequencing errors are introduced according to the probability distri-

bution implied by the base quality string. A user-specified fraction of these errors are produced as indel errors rather than substitution errors.

When simulating paired-end sequencing, the fragment length is modeled with a normal distribution. The user provides the mean and standard deviation for this distribution. Both reads are sampled from a single walk through the graph with length equal to a sampled fragment length. If the sampled fragment length is longer than the path it is being sampled from, the fragment length is truncated to the path length. If the sampled fragment length is shorter than the read length, the read is truncated to the fragment length.

3.6.5 Simulating RNA-seq reads from haplotype-specific transcripts

Reads were simulated from haplotype-specific transcript (HST) paths derived from the haplotypes of NA12878 in the 1000 Genomes Project (1000GP) and the GENCODE transcript annotation. The corresponding spliced pangenome graph (including the paths) was created using VG RNA. Identical HSTs were not collapsed, so that reads could be simulated from each haplotype independently.

In total, we created four different simulated read sets (Supplementary Table A.4): two sets each training with the SRR1153470 and ENCSR000AED read sets. For each training data set, one set of reads was simulated with an expression profile derived from the training data, and the other set was simulated with uniform expression across transcripts. The read sets with uniform expression were used to evaluate mapping, whereas the sets with data-based expression were used to evaluate expression inference. For the simulated read sets with data-derived expression, the reads were first mapped using Bowtie2 [107] with default parameters

and then expression-quantified using RSEM [114], also with default parameters. To ensure balanced expression between the two haplotypes for all transcripts, only transcripts that were successfully projected to both haplotypes were given a positive expression. The fragment length distribution mean and standard deviation estimated by RSEM was used to parameterize the fragment length distribution in VG SIM. For all four read sets, we simulated 25M 101 base-pair read pairs from each haplotype with an indel probability error of 0.001 and the base quality distribution trained on 10M randomly sampled read-pairs of the training data. The read-pairs were sampled using seqtk [116].

3.6.6 Mapping and multipath alignment with VG MPMAP

Like most read mappers, VG MPMAP's mapping algorithm is designed using the "seed-cluster-extend" paradigm. First, it locates exact matches "seeds" between the read and the graph. Next, the seeds are "clustered"' together to identify regions of the graph that the read could align to. Finally, the seeds are "extended" into an alignment of the entire read. Because these operations occur in the context of a pangenome graph, they use several specialized algorithms and indexes.

3.6.6.1 Seeding

VG MPMAP seeds alignments with maximal exact matches (MEMs) against the graph, which it finds using a GCSA2 index [189]. MEMs are exact matches between an interval of the read and a walk in the graph such that the match cannot be extended further in either direction at that location in the graph. The MEMs are found using a two-stage algorithm, which has also

been described previously [74].

In the first stage, the algorithm finds super-maximal exact matches (SMEMs), which are MEMs for which the read interval is not contained within the read interval of any other MEM (Supplementary Algorithm 1). This algorithm also relies on a longest common prefix (LCP) array, which allows navigation upward in the implicit suffix tree that the GCSA2 encodes. The second stage of the algorithm finds the minimally-more-frequent MEMs of each SMEM, subject to a minimum length (Supplementary Algorithm 3). These are the longest MEMs that are shorter than the SMEM but have their read interval contained in the SMEM's read interval. This stage also relies on the GCSA2 index.

3.6.6.2 Clustering

The clustering algorithm in VG MPMAP is built around the distance index described in [32]. In brief, this index can query the minimum distance between two positions in the pangenome graph by expressing the distance as the sum of a small number of precomputed distances. This is accomplished by taking advantage of the common topological features of pangenome graphs, namely that they tend to contain long chains of bubble-like motifs that result from genomic variation. These features are captured in the graph's "snarl decomposition", in which a snarl is one of these bubble-like motifs [154].

The clustering algorithm begins by constructing a directed acyclic graph (DAG) in which the nodes correspond to MEM seeds. The edges are added whenever 1) there is a path connecting two seeds in the graph, and 2) the seeds are collinear along the read. Note that the collinearity criterion guarantees acyclicity. We use the distance index to determine the

existence of a path that connects the seeds in the graph, and the edges are also labeled by the distance. Edges that are much longer than the read length are not added; this avoids treating distal elements on the same chromosome as part of the same cluster. In addition, we accelerate this process using Algorithm 3 from [32], which partitions seeds into equivalence classes based on the distance between them. The equivalence relation is the transitive closure of the relation of being connected by a path of length less than d, which is a tunable parameter. By choosing d correctly, we can ensure that all of the edges we would include occur between seeds in the same equivalence class. This significantly reduces the number of distance queries we need to perform.

Once the DAG of seeds has been constructed, we approximate the contribution of each seed and edge to the score of an alignment that contains them. Seeds are scored as if they are a short alignment of matches, and edges between seeds may be scored as an insertion or deletion if the distance in the graph does not match the distance on the read. These values serve as node and edge weights. We then use dynamic programming to compute the heaviest path defined by the node and edge weights within each connected component of the DAG and take the seeds along this path as a candidate cluster. Clusters are passed through to the next stage of the algorithm if their weight is within a prespecified amount of the heaviest-weight cluster, subject to a hard limit on the total number of clusters.

3.6.6.3 Multipath alignments

Most existing sequence-to-graph aligners, including VG MAP, produce an alignment of the sequence to a particular path through the graph. VG MPMAP uses a different alignment

formalism, which we call a multipath alignment. In a multipath alignment, the sequence can diverge and reconverge along different paths through the graph (Supplementary Figure A.1). Thus, the read can align to a full subgraph rather than to a single path. This allows the alignment object to carry within itself the alignment uncertainty at known variants or splice-junctions. This information can be used in downstream inference applications, including RPVG.

More formally, a multipath alignment of read R is itself a digraph with the following properties:

1. Each node corresponds to an alignment of some substring of R to a path in the pangenome
2. An edge between u and v exists only if u and v align adjacent substrings of R to adjacent paths in the pangenome.
3. Every source-to-sink path through the multipath alignment can be concatenated into a complete, valid alignment of R to a path in the pangenome.

It is worth noting that multipath alignments are acyclic by construction, since the nodes can be partially ordered by the read interval that they align. VG MPMAP additionally annotates each node's partial alignment with its alignment score. The alignment score of any particular sequence-to-path alignment expressed in the multipath alignment can be computed efficiently by simply adding the partial alignments scores along the path.

While sequence alignments have well-established optimization criteria, there is no such criterion for optimizing the topology of a multipath alignment. In lieu of one, we adopt heuristics that are motivated by the common topological features of pangenome graphs. Our

high-level strategy is to use exact match seeds to anchor alignments. We then use dynamic programming to align between seeds and within sites of variation in the graph, which we identify using the snarl decomposition of the pangenome graph. Using a multiple-traceback algorithm, we can then obtain alignments to different paths through the graph as necessary.

3.6.6.4 Anchoring alignments

To use a cluster of exact match seeds to anchor a multipath alignment, it is first necessary to compute the reachability relationships between the seeds. This is a non-trivial problem.

We begin by converting the local graph around a cluster into a directed acyclic graph using an algorithm that has been described previously [74]. In brief, we identify small feedback arc sets within each strongly-connected component using the Eades-Lin-Smyth algorithm [56], and then we duplicate the strongly-connected component with the feedback arcs linking successive copies. Using dynamic programming over the DAG as we construct it, we can preserve all cyclic walks up to some prespecified length, which is based on the read length.

After creating the DAG, we inject the seeds into the new graph. Since the DAG conversion algorithm can expand the node space of the original graph, seeds can now correspond to multiple locations in the DAG. In this case, we duplicate the seeds to all of the corresponding locations in the DAG. We then use a three-stage algorithm that computes the transitive reduction of a graph in which the nodes correspond to seeds, and two seeds have an edge between them if they are collinear along the read and reachable within the pangenome graph (Supplementary Algorithm 4).

1. Compute the reachability relationships between the seeds, ignoring collinearity on the

read.

2. Rewire the reachability edges between the seeds to respect collinearity on the read.

3. Compute the transitive reduction of the resulting graph.

This algorithm is designed to have linear run time in the number of seeds and the size of the DAG, but only in the typical case where the seeds line up along a walk through the pangenome graph. In the general case, the run time can be quadratic.

3.6.6.5 Dynamic programming with multiple traceback

The alignments between anchors (i.e. the vertices in the transitively reduced DAG) are computed using a banded implementation of partial order alignment [111]. The alignments of the read tails past the end of anchors are computed using a SIMD-accelerated POA implementation from the gssw library.

We use a specialized traceback algorithm to obtain the alignments to multiple paths through the pangenome graph from a single dynamic programming problem (Supplementary Algorithm 8). Instead of the optimal alignment, the algorithm returns the k highest-scoring alignments. We choose k to be the number of paths through the subgraph we are aligning to, subject to a hard maximum. The key insight behind the algorithm is that the next highest-scoring traceback can be determined by checking local properties of the dynamic programming matrix while computing the highest-scoring traceback. In addition, for each anchor that crosses a snarl, we remove the interior of snarl before performing alignments. This way, the multiple traceback algorithm can align to multiple paths at sites of variation.

3.6.6.6 Quantifying mapping uncertainty

The method that VG MPMAP uses to compute mapping quality is largely shared with VG MAP (see [74] Supplementary Note). As in VG MAP, base qualities are incorporated into alignment scores (essentially downweighting low-quality bases), and the alignment scores are subsequently used to compute a mapping quality. The formulas used to compute mapping quality rely on the conversion of alignment scores into the log-likelihood of a hidden Markov model (HMM), as described in [55] and [98].

VG MPMAP also uses a concept of a mapping's "multiplicity" to model errors introduced by the mapping algorithm itself. In particular, at certain points in the algorithm, we enforce hard caps on certain algorithmic behaviors, such as the number of alignments that will be attempted, in order to prevent excessive run time. If we run up against these hard caps, we expect that not all high-scoring alignments will be found. We incorporate this information into the mapping quality formula by treating alignments as if multiple equivalent alignments actually were found. For example, if we attempted alignments for 10 of 30 promising clusters and found 1 high-scoring alignment, we would estimate its multiplicity to be 3. That is, we estimate that there are a total of 3 alignments that are equally high-scoring, including the ones that we did not find. We then compute the mapping quality as if 2 additional copies of the alignment had been found.

Multiplicities allow VG MPMAP to aggregate information about sources of algorithmic inaccuracy over different steps in the algorithm. The central entities in each step of the mapping algorithm (seeds, clusters, alignments, and pairs) are each associated with a multi-

plicity. These multiplicities follow particular combining rules between successive steps of the algorithm. When combining orthogonal pieces of information (seeds in a cluster, or single-end alignments in a paired alignment), the new entity receives the minimum of its constituents' multiplicities. When layering on a new source of algorithmic uncertainty (typically a further hard cap), an entity's multiplicity is multiplied by its estimated multiplicity in that step of the algorithm.

3.6.6.7 Determining statistical significance

VG MPMAP uses a frequentist hypothesis test to assess the statistical significance of a read alignment. The test statistic that we use is the alignment score. The null hypothesis is that the alignment score was obtained by a uniform random sequence of the same length as the read. By default, we set the type-I error rate to 0.0001. If an alignment score's p-value is not significant at this level, the alignment is still reported, but its mapping quality is set to 0.

Modeling the null hypothesis of the test is not entirely straightforward. In general, we expect higher local alignment scores from longer reads or larger pangenome graphs. However, there are subtleties. A large pangenome graph may consist of many repeats of the same sequence so that its effective size is smaller than its total sequence length. Alternatively, a small pangenome graph may have a complex topology that admits a combinatorially large set of walks. For these reasons, we take an empirical approach that fits a model to match the pangenome graph. At the start of every mapping run, we map a sample of uniform random sequences of varying lengths. The resulting alignment scores are used to fit the parameters of a distribution using maximum likelihood, and those parameters are regressed against the read

length. The regression allows us to query the p-value for a read of any length.

The parametric distribution we use can be derived as the maximum of ν independent, identically distributed (i.i.d.) exponential variables with rate λ. This distribution has the following probability density function:

$$f(x|\lambda,\nu) = \lambda\nu(1-e^{-\lambda x})^{\nu-1}e^{-\lambda x}. \tag{3.1}$$

The fitting algorithm alternates between maximizing the likelihood with respect to each of the two parameters with the other fixed until convergence. ν is fit using the Newton-Raphson method, and λ is fit using golden-section search.

The motivation for this model is that the length of the match starting at position of a uniform random sequence (the read) and position of a fixed sequence (the reference) is approximately Geometric$(1/4)$, assuming the two sequences are relatively long. The optimal local alignment score is closely related to the longest match at any position on the read sequence to any position on the pangenome graph. Moreover, most of these matches have only weak dependence on each other, so the i.i.d. approximation is reasonable. We use an exponential distribution because it closely approximates a geometric distribution and is easier to fit.

3.6.6.8 Paired-end mapping

VG MPMAP has several features designed to take advantage of the paired-end sequencing reads produced by Illumina sequencers. At the beginning of each paired-end mapping run, VG MPMAP uses a sample of the first 3,000 uniquely mapped pairs to fit parameters of a frag-

ment length distribution. The distance between the reads in each pair is computed with the distance index. Non-uniquely mapped pairs are buffered and then remapped after the fragment length distribution has been fit.

The fragment distribution is modeled as a normal random variable with mean μ and variance σ^2. We use a method of moments estimator for a truncated normal distribution so that the parameter estimation is robust to possible mismappings. In particular, we discard the largest and smallest $\frac{1-\gamma}{2}N$ fragment length measurements (default $\gamma = 0.95$). This procedure makes the estimator insensitive to a sufficiently small fraction of outliers. The remaining γN measurements correspond to a sample from a truncated normal distribution with the same μ and σ^2. The following estimators can be derived using method of moments on this truncated normal distribution:

$$\begin{aligned} \hat{\mu} &= \bar{x} \\ \hat{\sigma}^2 &= s^2 \left(1 - \frac{2\alpha\phi(\alpha)}{\gamma}\right)^{-1}, \end{aligned} \tag{3.2}$$

where $\bar{x}$ and s^2 are the empirical mean and variance among the retained γN measurements, and $\alpha = \Phi^{-1}\left(\frac{1-\gamma}{2}\right)$ is the left truncation point on a standard normal distribution.

When mapping paired-end reads, the clustering stage of the algorithm adds an additional step. First, each read in the pair's seeds are clustered as in the single-end algorithm. Next, the clusters from the two reads are paired by checking which pairs imply a fragment length within 10 standard deviations of the mean, as estimated by the algorithm in the previous section. The implied fragment length connecting two clusters is estimated using the distance index, with the position of a cluster taken to be the position of its longest seeds. Pairs of clusters

are prioritized by a sum of an estimated alignment score (interpreted as a log-likelihood) and the log-likelihood of the normal distribution that we model the fragment length distribution with.

The heuristics used for read mapping inevitably fail in some cases. When mapping paired-end reads, it sometimes happens that the heuristics fail on only one of the two reads of a fragment. When this occurs, it is sometimes possible to "rescue" the alignment of the other read by aligning it to the region of the pangenome graph where we expect to find it relative to the mapped read.

VG MPMAP employs this strategy whenever the pair clustering procedure fails to produce a pair of clusters consistent with the fragment length distribution, or when all of the clustered alignment pairs have at least one end without a statistically significant alignment. We also perform a limited number of rescues even when a consistent cluster pair is found, provided that there are clusters of at least one of the ends that are equally as promising as the one in the cluster pair. This helps improve the calibration of mapping qualities. We place a hard cap on the number of rescues performed to control run time. The fraction of eligible rescues that were actually performed becomes a component in the multiplicity of an alignment, as described previously.

The multipath alignment algorithm is slightly different when computing rescue alignments. This is necessary because there are no exact match seeds to use as anchors. Instead, we first perform a single path alignment using gssw. Then we remove any sections of the alignment that lie inside snarls, and realign those segments of the read as when connecting anchors in the standard multipath alignment algorithm.

3.6.6.9 Spliced alignment

Because spliced pangenome graphs include annotated splicing events as edges, it is usually unnecessary to use specialized alignment algorithms to obtain spliced alignments. However, even for well-annotated genomes, transcript annotations are incomplete, especially for lowly-expressed transcripts. Thus, it is still important to be able to produce spliced alignments. VG MPMAP includes a spliced alignment algorithm but applies it conservatively: only when the primary alignment includes a long soft-clip on at least one end. A long soft-clip is suggestive that the clipped end of the read might align to a part of the graph that was too distant to be included in the primary seed cluster, as would be expected with an unannotated splice event.

The spliced alignment algorithm begins by finding candidate regions to align the clipped read end to. These regions are selected by scanning over secondary mappings, unaligned seed clusters, and unclustered seeds. To be considered, they each must 1) roughly correspond to the clipped end of the read, and 2) be reachable from the primary alignment by some path in the graph. Reachability is determined using the minimum distance index. Candidates are excluded if they are too distant from the primary alignment (default 500 kbp).

Next, the spliced alignment algorithm looks for splice motifs near the ends of the pair of splice candidates. If any pair of canonical splice site dinucleotides are found on any path from the two ends, the intervening sequence is aligned as if the two splice sites were joined by an edge in the graph. Splice motifs are penalized by their log-frequency, as given by Burset, et al. [25]. A spliced alignment is deemed to be statistically significant if the increase in score relative to the unspliced primary alignment would have been sufficient to be a statistically

significant mapping for the entire read. The spliced alignment algorithm is repeated until no statistically significant spliced alignments are discovered.

3.6.7 RNA-seq mapping evaluation

We compared VG MPMAP's performance at mapping RNA-seq data against the vg toolkit's existing graph alignment method VG MAP [74] and two state-of-the-art RNA-seq mapping tools, HISAT2 [103] and STAR [54]. Graph indexes and genomes were created for each tool using default parameters, with MPMAP and MAP sharing the XG and GCSA index. The mapping compute and memory usage of each tool were estimated using 16 threads on an m5.4xlarge AWS instance. All mappers were run with default or recommended parameters for RNA-seq data. The SRR1153470 and CHM13 data were used to optimize the parameters of VG MAP and VG MPMAP.

We evaluated mapping accuracy on simulated reads using two different methodologies to ensure the robustness of our conclusions. One methodology was based on basewise overlaps along the linear reference genome, and the other was based on distances along transcript and reference paths in the graph.

For the overlap-based evaluation the graph alignments were first projected to the reference paths using VG SURJECT in spliced alignment mode. Briefly, SURJECT takes a set of graph-aligned reads and re-aligns them to all nearby reference paths in the graph, producing a BAM file with the reads aligned to the reference sequences. The re-alignment is only performed on the parts of the alignment that do not already follow the reference paths. A read

was considered correctly mapped if 70% or 90% (depending on the evaluation) of the bases of the simulated true reference alignment were covered by the estimated alignment. The true reference alignments were generated using the transcript position of each read provided by VG SIM and the NA12878 haplotype-specific transcript reference alignments. The latter were created by projecting the transcript paths to the reference sequences using VG SURJECT in spliced alignment mode. Due to sequencing artifacts, the ends of reads will occasionally consist of such low-quality bases as to be practically random. Aligners that decide to softclip these uninformative bases would be penalized in this overlap-based evaluation. We therefore decided to trim all bases at both ends of an alignment (including the true alignments) that had a phred base quality score below 3. All alignments for which more than half of the sequence was trimmed were discarded from the evaluation so that the percent overlap could be estimated more confidently.

We used the VG GAMPCOMPARE tool for the distance-based evaluation. The truth set in this evaluation was the true graph alignments produced by VG SIM. In short, VG GAMPCOMPARE finds the minimum possible distance between the start position of an estimated alignment and the true alignment across all reference and transcript paths in the graph. Before running GAMPCOMPARE, HISAT2 and STAR's BAM format alignments were converted into graph alignments (GAM format) using VG INJECT, which translates linear reference alignments into alignments against the path of the reference in a graph. An alignment was considered correct if its start position was within 100 bp of the start position of the true alignment along the path of the reference or any transcript path.

Reference bias was quantified using simulated reads, by counting the number of reads that

overlapped variants with a mapping quality value of at least 30. For this analysis we used the reference-based alignments (projected alignments for VG MAP and VG MPMAP). In order to treat different variant types and lengths equally, we computed the read count for each variant as the average read count across the variant's two breakpoints. Reads simulated from each haplotype were counted separately and only variants with at least 20 reads across both alleles combined were used to quantify reference bias. Complex variants that were not classified as SNVs, simple deletions or simple insertions were skipped.

When benchmarking using real reads, truth alignments are not available. Instead, we used a proxy measure of aggregate mapping accuracy based on long read mappings from the same cell line. The long reads are easier to map confidently, and we expect the cell line to have similar transcript expression across replicates. Thus, higher correlation between the coverage of short read mappings and the coverage of long read mappings is suggestive of higher accuracy. For long read data, we used NA12878 PacBio Iso-Seq alignments generated by the ENCODE project (Supplementary Table A.3). The cleaned Iso-Seq alignments of four replicates were first merged and secondary alignments and alignments with a quality below 30 were filtered using SAMTOOLS [121]. These filtered alignments were then compared to the short-read RNA-seq alignments by calculating the Pearson correlation of the average exon read coverage between the two. Exons were defined using the Iso-Seq alignments by first converting them to BED format and then merging overlapping regions using BEDTOOLS [165].

We measured memory and compute time for all mappers using the Unix TIME utility. The reads per second statistic was computed by dividing the number of reads by the product of the wall clock time and the number of threads. This is a somewhat biased measurement, since it includes the one-time start up computation that does not scale with the number of reads. However, the magnitude of this bias is small, and it tends to disfavor VG MPMAP, which has the longest start up of the tools we evaluated.

All secondary alignments were filtered in all evaluations using SAMTOOLS. Reference alignments in BAM format were sorted and indexed, also using SAMTOOLS. The SeqLib library was used in the evaluation scripts to parse the alignments and calculate overlaps [212].

3.6.8 Haplotype-specific transcript quantification

We developed RPVG as a general tool for inferring the most likely paths and their abundance from a set of mapped sequencing reads. In this study we used RPVG to quantify the expression of haplotype-specific transcripts (HSTs) in a pantranscriptome. RPVG's algorithm consist of four main steps:

1. Find read alignment paths that align to HST paths
2. Cluster alignment paths and HST paths
3. Calculate alignment path probabilities
4. Infer haplotypes and expression from probabilities

A graphical overview can be seen in Supplementary Figure A.11.

3.6.8.1 Finding alignment paths

The first step of RPVG is to parse each alignment and find all alignment paths that align to (i.e. follow) at least one HST path in the pantranscriptome GBWT index (Supplementary Figure A.11a). An alignment path is the set of nodes a read alignment follows in the graph. For single-path alignments there is only one alignment path, but for multipath alignments there can be many. We will here focus on multipath alignments, since a single-path alignment is merely the simpler case when a multipath alignment only contains a single path.

Multipath alignments are represented as a graph, and thus the objective is to find all paths through this graph that also exist as subpaths in the GBWT. In other words we want to find all possible alignments that the read can have to all HST in the pantranscriptome. This search would normally scale linearly in the number of HSTs overlapping the read, but the GBWT allows us to query all HSTs that contain the same subpath. Therefore, HSTs that are locally identical will be queried together, taking advantage of the fact that haplotypes are markedly more similar locally than globally.

RPVG uses a depth-first-search (DFS) through the multipath alignment graph to find all alignment paths. A branch in the search is terminated if its alignment path is not present as a subpath in the GBWT. A DFS is initialised at each source node in the alignment graph. We terminate any alignment path early where it is not possible to reach a score of 20 below the current highest scoring path, assuming perfect scoring for the remainder of the alignment.

The topology of the multipath alignment graphs is determined by heuristics. In some

cases these heuristics fail, resulting in multipath alignments that do not cover all possible alignment paths. This can result in incorrect downstream expression estimates as a read might be missing an alignment path to the correct HST. To overcome this, RPVG allows alignment paths to be shortened in order to be made consistent with an HST path. More specifically, the DFS can start and end up to four bases inside the read (excluding soft-clipped bases). The score of partial alignment paths are penalized proportionally to the number of non-matched bases at each end, adjusted for their quality.

The output from the DFS is one set of alignment paths for each multipath alignment. Next, RPVG labels a set as low scoring if the highest scoring alignment path in the set is less than 0.9 times the optimal quality-adjusted alignment score, which is the score an alignment would get if it consisted of only matches. The sets labeled as low scoring are treated as being incorrect; they may be misalignments, or originate from an HST not in the input pantranscriptome. The labeled sets are later used when calculating the noise probability.

For paired-end reads, one additional step is needed: combining the alignment paths of each read to create a set of alignment paths for the whole fragment. First a set of alignment paths is generated for each alignment in the pair as described above. Next, RPVG attempts to combine each start (first read) alignment path with each of the end (second read) alignment paths. If the fragments are not strand-specific and the pantranscriptome GBWT is not bidirectional, RPVG then repeats the process using the reverse complement of the fragment.

The procedure to combine the two alignment paths differs depending on whether they overlap or not. If they do overlap, a single combined alignment path is created for the fragment

by merging the two while requiring that the path of overlapping portions matches perfectly. If they are separated by an insert, the start alignment path is extended using a DFS following the HST paths. If the search reaches one of the start nodes for an end alignment path, a new fragment alignment path is created by merging the search and end alignment path. The new fragment alignment path is only kept if it follows at least one HST path in the pantrancriptome. The search is terminated if all start nodes in the end alignment paths have been visited and they are not part of a cycle. An alignment path is discarded if its length is above $\mu + 5\sigma$, where μ and σ are the mean and standard deviation of the fragment length distribution. These parameters are either supplied by the user or parsed from the input alignments (the VG aligners write the parameters they estimated to the alignment file). The score of the resulting fragment alignment path is calculated as the sum of the scores of the two read alignment paths. The mapping quality is calculated as the minimum across the two reads.

The final output from the search is a set of alignment paths and the HSTs that each path aligns to for each read or fragment. For simplicity, in the following, we will use the term "fragment" to denote both a single-end read and a set of paired-end reads.

3.6.8.2 Clustering transcript paths

HST paths that do not share any fragments are independent, and therefore their expression can be inferred separately. In contrast, the expression of HST paths that share alignments must be inferred jointly. Accordingly, RPVG identifies clusters of HST paths that share alignment paths from the same fragment. By dividing the inference problem into these smaller,

independent clusters, computation and memory can be considerably reduced.

The clustering algorithm works by first constructing an undirected graph where vertices correspond to HST paths and edges correspond to HST paths being observed in the same set of fragment alignment paths. All connected components in this graph (clusters) are then located using breadth-first-search. All fragments are assigned to their respective clusters based on the HST paths that their alignment paths align to.

3.6.8.3 Calculating alignment path probabilities

For each fragment, the probability of it originating from each of the HSTs in its cluster is calculated by RPVG using the alignment path scores, lengths and mapping quality (Supplementary Figure A.11b). First the probability ε that the fragment was not from any of the HST in the cluster is calculated using the mapping quality q:

$$\varepsilon = \max\left(\varepsilon_{min}, 10^{-q/10}\right), \tag{3.3}$$

where ε_{min} is the minimum noise probability. The motivation behind having a minimum is that mapping qualities are generally less reliable at higher values. The minimum noise probability is 10^{-4} for all fragments except those that were labeled as low scoring, for which it is 1. Now, let A be the set of alignment paths (i.e. alignments) for this fragment. For each alignment path $a \in A$, the likelihood of it being the correct path is calculated using its score s_a and length ℓ_a:

$$L(a) = \phi\left(\frac{\ell_a - \mu}{\sigma}\right)\exp\left(\lambda s_a\right), \tag{3.4}$$

where ϕ is the standard normal density function, λ is a scaling factor that converts the alignment score into the log-likelihood of a pair-HMM [98], and μ and σ are the mean and standard deviation of the fragment length distribution modeled using a normal distribution. For paired reads, these parameters are estimated from the alignment path lengths across all fragments that have 1) a mapping quality of at least 30, and 2) the same length for all alignment paths. The fragment length distribution is omitted from the equation when the fragments are single-end reads. With this likelihood, we can compute the posterior probability that the fragment originated from a given HST. Let the set of all HST paths in the cluster be denoted by T, and let the set of HST paths an alignment path a is consistent with be denoted by T_a. The probability that the fragment (or alignment A) originated from an HST is calculated as:

$$p_t = (1-\varepsilon) \cdot P(t|A) = (1-\varepsilon) \cdot \frac{P(A|t)P(t)}{\sum_{t \in T} P(A|t)P(t)} \tag{3.5}$$

with

$$P(A|t) \propto \max_{a \in A} \begin{cases} \frac{L(a)}{\widetilde{\ell}_t} & \text{if } t \in T_a \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Here, $\widetilde{\ell}_t$ is the effective transcript length for t calculated as $\widetilde{\ell}_t = \ell_t - \mu_{\ell_t}$. In turn, μ_{ℓ_t} is the mean of the fragment length distribution truncated to $[1, \ell_t]$. A similar approach is used in SALMON [157]. The effective transcript length accounts for the fact that fragments cannot be sequenced from all positions due to the size of the fragment. If the fragments are single-end reads, the fragment length distribution parameters used to calculate the effective length must be

supplied by the user. The prior over HSTs $P(t)$ is taken to be uniform. If the HST probability p_t is below 10^{-8}, it is truncated to 0 to reduce storage.

We denote the set of all fragment probabilities in a cluster as F and the probabilities for a fragment i as $F_i = (\varepsilon, \mathbf{p})$, where $\mathbf{p}$ is the vector of probabilities over all T HSTs in the cluster. Many fragments will have very similar probabilities and can thus be collapsed to save computation resources and memory [147, 157]. To do this we collapse two fragment probabilities F_i and F_j if they satisfy both of:

$$\begin{aligned} \left|\varepsilon^i - \varepsilon^j\right| &< 10^{-8} \\ \left|p_t^i - p_t^j\right| &< 10^{-8}, \quad \forall t \in T \end{aligned} \tag{3.7}$$

We also associate each set of collapsed fragments with c, the number of collapsed fragments in the set. The resulting set E of tuples $(\varepsilon, \mathbf{p}, c)$ is subsequently used to infer the expression of the HSTs in the pantranscriptome.

3.6.8.4 Inferring haplotype-specific transcript expression

RPVG quantifies the expression of the HSTs in the pantranscriptome using a nested inference scheme (Supplementary Figure A.11c). This is done independently for each cluster. First, the distribution over diplotypes (i.e. pairs of haplotypes) is inferred. A haplotype combination is then sampled from this distribution and expression is inferred conditioned on the sampled haplotypes. This procedure is repeated multiple times to account for the uncertainty in the haplotype estimates. In the following, we will assume the sample is diploid, but the equations and algorithms generalize to any ploidy.

The marginal distribution over diplotypes is approximated by assuming the haplotypes are identical for all transcripts in a cluster. The motivation behind this approximation is that most clusters cover only a small region (e.g. gene) of the genome. However, this approximation can break down when there are partial haplotypes or recombination events in the cluster. Using the transcript and haplotype origin table provided by VG RNA, the HSTs in the cluster are first grouped by their haplotype origin. Note that since an HST can be consistent with more than one haplotype it can also belong to multiple groups. Next, groups with the same set of HSTs are collapsed resulting in a set of unique haplotype groups.

Now let us denote the set of haplotype groups as H, with each group $h \in H$ consisting of a set of HSTs. The objective is to infer the distribution over diplotypes $d = \{h_1, h_2\}$ conditioned on the set of collapsed fragment probabilities E. The probability of a diplotype is defined as:

$$P(d|E) = P(\{h_1,h_2\}|E) \propto P(h_1)P(h_2) \prod_{(\varepsilon,\mathbf{p},c)\in E} \left(\varepsilon + \frac{1-\varepsilon}{2}\left(P(\mathbf{p}|h_1) + P(\mathbf{p}|h_2)\right)\right)^c \tag{3.8}$$

and

$$P(\mathbf{p}|h) = \frac{\frac{1}{n}\sum_{t\in h} p_t}{\sum_{k\in H}\frac{1}{n}\sum_{t\in k} p_t} \propto \sum_{t\in h} p_t \tag{3.9}$$

where the prior probability of each haplotype group $P(h)$ is proportional to the number of haplotypes in the group, and n is the number of transcripts in the cluster ($\frac{1}{n}$ and $\frac{1}{2}$ amount to

an approximation that expression is uniform across all transcripts and the two haplotypes, respectively). This model is inspired by similar haplotyping models used in Platypus and other genotypers [3, 161, 174].

The distribution over diplotypes is inferred by calculculating $P(d|E)$ for all pairs of haplotype groups $h \in H$. To reduce the space of haplotype combinations that need to be evaluated, RPVG uses a branch-and-bound-like algorithm, where diplotypes containing an improbable haplotype group are not evaluated. Instead, the probability of all diplotypes containing an improbable haplotype group is set to 0. A haplotype group h is labeled to be improbable if its optimal diplotype probability $P(\{h, h_o\}|E)$ is $s \cdot 10^4$ times lower than the current highest evaluated probability, where s is the number of diplotypes sampled in the next step in the inference. The optimal diplotype probability is defined as

$$P(\{h, h_o\}|E) \propto P(h) \prod_{(\varepsilon, \mathbf{p}, c) \in E} \left(\varepsilon + \frac{1-\varepsilon}{2} \left(P(\mathbf{p}|h) + \max_{h_o \in H} \left(P(\mathbf{p}|h_o) \right) \right) \right)^c \tag{3.10}$$

This value serves as an upper bound on the probability of any diplotype containing h.

Using the inferred distribution over diplotypes the expression of the HSTs in the cluster is inferred. First a diplotype is sampled from the distribution $P(d|E)$ and all HSTs that are consistent with at least one of the haplotypes in the diplotype are collected. We denote this HST subset $T_s \subseteq T$ and define the likelihood over the expression values α as

$$L(\alpha) = \prod_{(\varepsilon, \mathbf{p}, c) \in E} \left(\sum_{t \in T_s} \alpha_t p_t \right)^{c \cdot \varepsilon}, \tag{3.11}$$

where $c_{\text{-}\varepsilon}$ is the noise-adjusted fragment count: $c_{\text{-}\varepsilon} = c(1-\varepsilon)$. To find the (local) maximum likelihood estimate of the expression values a expectation maximization (EM) algorithm is used. The algorithm iterates between assigning fractional fragment counts to the HSTs and updating the expression values. This is a well known algorithm that is used by many other transcript quantification tools [23, 114, 147, 157]. The expression values are initialized uniformly and the EM algorithm is run until convergence or for a maximum of 10,000 iterations. The algorithm is considered converged if

$$\frac{\left|\alpha_t^i - \alpha_t^{i-1}\right|}{\alpha_t^i} \leq 0.001, \quad \forall t \in T_s : \alpha_t \geq 10^{-8} \tag{3.12}$$

for 10 consecutive iterations, where i is the index of the current iteration. This criteria is inspired by the one used by KALLISTO [23] and SALMON [157]. All expression values below 10^{-8} of the maximum likelihood estimate are truncated to 0. The diplotype sampling and EM steps are repeated 1,000 times to propagate the uncertainty over diplotypes into the HST expression estimates. Since the EM algorithm is deterministic, the same expression values are inferred for the same diplotype. We therefore only need to run the EM step once for each uniquely sampled diplotype, which can considerably reduce computation time.

The final output of RPVG is the haplotype probability and estimated expression value for each HST in the pantrancriptome. The probability is calculated as the fraction of diplotype samples which included the HST. The expression reported is the average across all diplotype samples (including samples where the HST's expression is zero due to its haplotype being absent from

the diplotype).

3.6.9 Transcript quantification evaluation

We compared RPVG's quantification accuracy against three other transcript quantification tools: KALLISTO, SALMON and RSEM. Haplotype-specific transcript indexes for KALLISTO, SALMON and RSEM were built from the HST sequence FASTA files generated by VG RNA. SALMON indexing was run with duplicates kept and, on the real data, the reference genome was given as a decoy. The Bowtie2 mapper was used in RSEM with the maximum number of alignments per read increased to 1,000. The transcript expression was estimated using default parameters for all methods, except for the real data where strand-specific inference was enabled. KALLISTO and SALMON were run without bias correction as it did not provide a clear advantage on the "Europe (excl. CEU)" pantranscriptome using the SRR1153470 reads (data not shown). RSEM was only run on the NA12878 sample-specific transcriptome and the "Europe (excl. CEU)" pantranscriptome, as it did not scale to the two largest pantranscriptomes.

RPVG was run using default parameters and with two different types of alignments inputs: the standard multipath alignments from VG MPMAP and single-path alignments generated by finding the best scoring path in the multipath alignments using VG VIEW. The fragment length distribution parameters estimated by VG MPMAP were given as input to RPVG when using the single-path alignments as they are lost from the alignment file during the conversion. RPVG was run with a ploidy of 2 for all read sets, including CHM13. For the simulated data, the exon-only splicing graphs were used when mapping the reads using VG MPMAP. These graphs are more comparable to the transcriptomes that the other methods use for (pseudo)-alignment.

For the real data, we used the whole spliced pangenome graphs. All HSTs with a haplotype probability below 0.8 were filtered from the RPVG output. The SRR1153470 and CHM13 read data was used to optimize the parameters of RPVG.

For the SRR1153470 and ENCSR000AED data, which are both NA12878 cell lines, we compared the quantified HSTs to the NA12878's haplotypes from the 1000 Genomes Project data. We considered an HST consistent with these haplotypes if it matched the sequence of one of the two possible NA12878 haplotype versions of the transcript. The haplotyping performance of each method was then estimated by comparing the number and fraction of quantified HSTs with positive expression that were consistent.

We used transcripts per million (TPM) to measure expression. For the simulated data we re-calculated the TPM value for all methods. The reason was that we wanted to ensure that there was no bias towards RSEM, which was used to estimate the expression profile employed by VG SIM to parameterize the HST expression values. The TPM value depends on the effective transcript length, which is not calculated in the same manner for each method. Therefore, if this is not corrected, methods that estimate the effective transcript length more similarly to RSEM will have an advantage that does not depend on their ability to predict correct expression values. The true fragment length distribution parameters and the effective transcript length approach employed by RPVG (similar to SALMON) was used when re-calculating the TPM values.

The method's ability to predict the correct expression value was evaluated using the simulated data for which the true expression is known. The true expression values were calculated from a table provided by VG SIM, which indicates the transcript of origin for each read.

The simulated TPM values were calculated in the same manner as described above. We used both Spearman correlation and mean absolute relative difference (MARD) to quantify concordance between estimated and true expression.

The CHM13 cell line is effectively haploid, so only a single HST is expected to exist for each transcript. We used this feature of the data to measure the haplotype inference performance of each method on the T2T CHM13 data. We defined each HST as either major or minor. Major HSTs were defined as the highest expressed haplotype for each transcript; the rest were defined as minor. The fraction of expression from minor HSTs is a lower bound on the fraction of incorrectly inferred transcript expression. Accordingly, we used the number of major and minor transcripts that each method predicted to be expressed to compare their haplotype inference performance.

3.6.10 Transcript quantification evaluation

We compared RPVG's quantification accuracy against three other transcript quantification tools: KALLISTO, SALMON and RSEM. Haplotype-specific transcript indexes for KALLISTO, SALMON and RSEM were built from the HST sequence FASTA files generated by VG RNA. SALMON indexing was run with duplicates kept and, on the real data, the reference genome was given as a decoy. The Bowtie2 mapper was used in RSEM with the maximum number of alignments per read increased to 1,000. The transcript expression was estimated using default parameters for all methods, except for the real data where strand-specific inference was enabled. KALLISTO and SALMON were run without bias correction as it did not provide

a clear advantage on the "Europe (excl. CEU)" pantranscriptome using the SRR1153470 reads (data not shown). RSEM was only run on the NA12878 sample-specific transcriptome and the "Europe (excl. CEU)" pantranscriptome, as it did not scale to the two largest pantranscriptomes.

RPVG was run using default parameters and with two different types of alignments inputs: the standard multipath alignments from VG MPMAP and single-path alignments generated by finding the best scoring path in the multipath alignments using VG VIEW. The fragment length distribution parameters estimated by VG MPMAP were given as input to RPVG when using the single-path alignments as they are lost from the alignment file during the conversion. RPVG was run with a ploidy of 2 for all read sets, including CHM13. For the simulated data, the exon-only splicing graphs were used when mapping the reads using VG MPMAP. These graphs are more comparable to the transcriptomes that the other methods use for (pseudo)-alignment. For the real data, we used the whole spliced pangenome graphs. All HSTs with a haplotype probability below 0.8 were filtered from the RPVG output. The SRR1153470 and CHM13 read data was used to optimize the parameters of RPVG.

For the SRR1153470 and ENCSR000AED data, which are both NA12878 cell lines, we compared the quantified HSTs to the NA12878's haplotypes from the 1000 Genomes Project data. We considered an HST consistent with these haplotypes if it matched the sequence of one of the two possible NA12878 haplotype versions of the transcript. The haplotyping performance of each method was then estimated by comparing the number and fraction of quantified HSTs with positive expression that were consistent.

We used transcripts per million (TPM) to measure expression. For the simulated data

we re-calculated the TPM value for all methods. The reason was that we wanted to ensure that there was no bias towards RSEM, which was used to estimate the expression profile employed by VG SIM to parameterize the HST expression values. The TPM value depends on the effective transcript length, which is not calculated in the same manner for each method. Therefore, if this is not corrected, methods that estimate the effective transcript length more similarly to RSEM will have an advantage that does not depend on their ability to predict correct expression values. The true fragment length distribution parameters and the effective transcript length approach employed by RPVG (similar to SALMON) was used when re-calculating the TPM values.

The method's ability to predict the correct expression value was evaluated using the simulated data for which the true expression is known. The true expression values were calculated from a table provided by VG SIM, which indicates the transcript of origin for each read. The simulated TPM values were calculated in the same manner as described above. We used both Spearman correlation and mean absolute relative difference (MARD) to quantify concordance between estimated and true expression.

The CHM13 cell line is effectively haploid, so only a single HST is expected to exist for each transcript. We used this feature of the data to measure the haplotype inference performance of each method on the T2T CHM13 data. We defined each HST as either major or minor. Major HSTs were defined as the highest expressed haplotype for each transcript; the rest were defined as minor. The fraction of expression from minor HSTs is a lower bound on the fraction of incorrectly inferred transcript expression. Accordingly, we used the number of major and minor transcripts that each method predicted to be expressed to compare their haplotype inference

performance.

3.6.11 Demonstration of analyzing genomic imprinting

We obtained RNA-seq data sets from samples NA11832, NA11930, NA12775, and NA12889 from Geuvadis data portal and ran them through the VG MPMAP-RPVG pipeline. Each sample had two accessions, which were combined into one data set (see Supplementary Table A.3). We also obtained and analyzed data from sample GM12878 from the ENCODE Project (ENCSR000AED replicate 1) [44]. These samples are all unrelated. All parameters used were identical to those used in the real data evaluations of VG MPMAP and RPVG above. The Geuvadis samples were used to troubleshoot the analysis and identify potentially interesting genes to highlight in the demonstration. The analyses were then repeated on the ENCODE sample. This design reduces the risk of identifying noise as signal. Only the results final analysis are the ones reported in the Results section, but results were broadly consistent across all samples.

To confirm that the pipeline could detect previously known ASE, we looked for signatures of imprinting in the 20 genes with the most statistically significant parent-of-origin ASE in the study by Zink, et al. [220] (Supplementary Table 6). One of these genes, *RP11-69E11.4*, had since been removed from the GENCODE database, so we excluded it from the analysis. Zink's, et al. study analyzed ASE on individual SNVs. To make our results comparable to theirs, we translated RPVG's HST-based expression quantification into a corresponding variant allele-based expression quantification. To do so, we used a table of the HSTs that contain each variant in the pantranscriptome, which can be produced by VG RNA. The expression of each

allele was computed as the sum of the expression of each HST that contained the allele.

We decided to highlight the haplotype-specific expression of the NAA60 gene in depth because it consistently showed monoallelic expression for both haplotypes across different isoforms in the initial exploratory data sets. To identify the haplotype of origin for different HSTs, we compared the variants associated with each HST (using the table from VG RNA) to the sample's haplotypes from the 1000 Genomes Project VCF. Equal-tailed credible intervals were approximated using RPVG's Gibbs sampling method.

3.6.12 Code and data availability

A list of the versions used of each method is available as Supplementary Tables A.5 and **??**. All bash scripts with exact command-lines used to generate the results are available at the following GitHub repository: `https://github.com/jonassibbesen/vgrna-project-paper`. This includes log files, references to the Docker containers used to run the methods and links to the raw simulated sequencing data used in the evaluation. Furthermore, all created spliced pangenome graphs and pantranscriptome haplotype-specific transcript sets are available for download at the repository for use in other projects. All custom C++, Python and R scripts used for the evaluation and plotting are available at `https://github.com/jonassibbesen/vgrna-project-scripts`.

Part III

Algorithmic infrastructure for pangenomics

Chapter 4

Interfacing with the linear reference-based software ecosystem

4.1 Preamble

This section contains a description of a tool in the VG toolkit that I designed and implemented. It is unpublished, and I have no intention to publish it beyond this dissertation in the future. However, I consider it a sufficiently significant contribution to the field of pangenomics to warrant its own chapter here. It has already been used as a component in a number of publications [47, 79, 80, 133, 191].

4.2 Introduction and motivation

One of the most fundamental tasks in computational pangenomics is mapping and aligning sequencing reads to a pangenome graph. The VG toolkit itself has three separate

mapping algorithms (VG MAP [74], VG MPMAP [187], and VG GIRAFFE [191]), and other tools have also been published [103, 167, 173]. Internally, all of these tools align the read sequence to the sequence of some walk through the pangenome graph. In the case of the three VG mapping tools and GRAPHALIGNER [173], they also output these graph-based alignments for downstream analysis.

Graph-based alignments cannot be easily expressed in the conventional file format used for read mappings: SAM/BAM [121]. In addition to the base-level alignment, the graph alignments must also encode the alignment's walk through the graph. For this reason, new file formats have been adopted: the VG toolkit's GAM format and more recently MINIGRAPH's GAF format [120]. Both of these formats are similarly expressive, so I will not emphasize the distinctions between them.

While the graph-based alignments enable some novel downstream applications [86, 173], the broader ecosystem of computational genomics tools remains heavily invested in the SAM/BAM format. To interface with this ecosystem, VG needs a method to convert its graph-based alignments into linear alignments that can expressed in SAM or BAM. In this chapter, I describe the VG SURJECT algorithm, which fills this role.

4.3 Design and implementation of VG SURJECT

Converting a linear alignment into a graph-based alignment is a trivial and lossless operation. In VG, the path of the linear reference sequence is maintained as a labeled walk within the pangenome graph, so all that is required is to look up the walk through the graph

taken by the aligned subsequence of the linear reference. The reverse transformation is more challenging. If any part of a read's alignment lies off of the labeled walk of the reference sequence, there is no lossless conversion. The VG SURJECT algorithm is designed with the principle that, despite the inevitable loss, as much of the graph-based alignment should be preserved as possible. The algorithm attempts to realign only the off-reference portions of the alignment onto the reference path so that the end result is nearly-identical alignment that can be losslessly converted into a linear alignment.

4.3.1 Connection to the multipath alignment problem

VG SURJECT begins by identifying the segments of the alignment that follow the path of the reference sequence. In doing so, it can also identify the interval of the reference path that the remainder could be aligned to. These reference-overlapping segments must then be connected with intervening alignments to this interval of the reference. This bears a strong similarity to the algorithm developed for VG MPMAP (see section 3.6.6.4) for constructing multipath alignments (Supplementary Algorithm 4). There, the task was to identify the reachability relationships between exact match seeds in a directed acyclic graph (DAG), which would then be connected with intervening alignments. Here, the task is to connect portions of an aligned sequence to a subinterval of the reference sequence (which can be seen as a very constrained DAG). VG SURJECT takes advantage of this similarity and repurposes the same underlying algorithm for this rather different use case.

4.3.2 Spliced alignments

Special considerations must be made for spliced alignments of RNA-seq data, which include long unaligned intronic sequences. In the SAM/BAM formats, expressing these alignments is relatively simple. The CIGAR string can include an 'N' operation, which indicates unaligned sequence. Due to the CIGAR string's run-length encoding, a long unaligned sequence requires similar space to represent as a short one. VG SURJECT's algorithm described above does not handle these cases so gracefully. When aligning to a spliced pangenome graph, the splice junctions are present in the graph as edges. However, when converting to an alignment to the path of the reference as an intermediate, the alignment to the intronic sequence requires explicitly spelling out the full path of the intron. For long introns, this can be prohibitively slow.

Because of this challenge, VG SURJECT includes specialized features for converting transcriptomic alignments. Before any step in the algorithm that would look at the full intronic sequence, it first attempts to decompose the problem into smaller realignment problems that fall within exons. The exonic segments are identified using a DAG in which the nodes correspond to reference overlapping segments of the alignment, and edges indicate that the segments are collinear on both the read and the reference sequence. All transitive edges in this graph are removed as a preprocessing step. An edge in this graph is flagged as a splice junction if it meets several criteria:

1. All source-to-sink paths in the DAG use the edge. This ensures that it is safe to assume that the full surjected alignment will also use this edge.

2. The two alignment segments connected by the edge directly abut on the read sequence.

This ensures that the intervening alignment corresponds to a deletion of the reference sequence.

3. The deletion implied by 2. is at least 20 bp long.

4. There is an edge in the spliced pangenome graph connecting the aligned segments, which suggests that the graph alignment follows known splice junction.

If these conditions are met, VG SURJECT partitions the realignment problem on either side of this edge, maintaining the alignment to the intron implicitly. When performing the final conversion to SAM/BAM, the implied 'N' operations are added to the CIGAR string explicitly.

The latter three conditions to identify a splice edge are fairly straightforward to check. Condition 1 is not quite trivial. However, it can be determined using a variant of the forward-backward algorithm that counts walks through the DAG. If the nodes are indexed in topological order by $i = 1, \ldots, N$, then we can count walks using the following dynamic programming recursion:

$$f_i = \begin{cases} 1, & i \text{ is a source node} \\ \sum_{\text{edges } (j,i)} f_j, & \text{else} \end{cases} \tag{4.1}$$

The total number of walks can then be computed by summing f_i over sink nodes. The same computation can be computed in reverse topological order as well:

$$b_i = \begin{cases} 1, & i \text{ is a sink node} \\ \sum_{\text{edges } (i,j)} b_j, & \text{else} \end{cases} \tag{4.2}$$

With both of these dynamic programming problems completed, the number of walks that use a given edge (i, j) is equal to $f_i \cdot b_j$. To verify condition 1 above, it is sufficient to compare this number to the total number of walks computed after the forward pass.

Chapter 5

Memory-efficient dynamic sequence graphs

5.1 Preamble

This chapter consists of the entirety of the paper "Efficient dynamic variation graphs", which was published in *Bioinformatics* [62]. I share primary authorship on this paper with Adam M. Novak, and the paper was also a close collaboration with the last author, Erik Garrison. This paper describes and compares several implementations of sequence graphs. I am responsible for implementing HashGraph, PackedGraph, and part of XG. Erik Garrison is responsible for implementing ODGI and part of XG. Adam M. Novak helped design the data model and implement the Python bindings (with help from Cecilia Cisar). Erik Garrison and I wrote the majority of the paper.

5.2 Introduction

As increasingly many individuals have been sequenced from certain species, the field of **computational pangenomics** has emerged to analyze whole populations of genomes rather than individual genomes [40]. Much of the research in computational pangenomics has coalesced around graph-based approaches for representing populations of genomes [155]. Unlike conventional string-based representations, graph data structures can represent genomic variation like substitutions, insertions, deletions, and other more complex genomic events.

Graph-based data structures present new computational challenges. In addition to sequence, genome graphs must represent topology. Given the size of many genomes, this can be quite demanding on computer memory. However, the total information content in a genome graph is only incrementally more than the sequences of the pangenome. This suggests that significant memory savings should be possible. There is also significant impetus to make the graph data structures computationally efficient, since they are frequently the core data structure in pangenomics applications.

Early versions of the variation graph toolkit (VG) [74] have provided a cautionary tale of a naïve implementation. VG used full-width machine words as identifiers for graph elements, and stored the elements and graph topology in a set of hash tables. Loading the 1000 Genomes Project's variant set into the VG toolkit used to consume more than 300 GB of memory, which is $\sim$30 times as large as the serialized representation [70].

Although VG provided a memory-efficient representation of the graph (XG) that could be used during read mapping and variant calling, this representation did not allow for

dynamic updates to the graph. The dynamic implementation remained necessary for graph-modifying steps of VG pipelines, such as the original construction of the graph and augmenting the graph with novel variants. Some pipelines could be made feasible by breaking large graphs into connected components. However, this strategy reduces efficiency, and it is untenable for pangenome graphs that consist of a single component.

To overcome this limitation, we have developed three new graph genome data structures that are dynamic (allowing efficient updates and edits) and also memory-efficient for real world genome graphs. Here, we compare the performance of these data structures to the original VG representation and to XG, using a diverse collection of genome graphs obtained during our work in graphical pangenomics.

In addition to demonstrating the possibility of working with large, complex graphs in small amounts of memory, these implementations expose a common API based on the HANDLEGRAPH model described below. This model provides an interface to genome graphs, based on their fundamental elements, which is intended to be implementable atop a broad diversity of graph storage designs. The VG toolkit has been refactored to use this API as its default means of loading, saving, and manipulating graphs since version 1.22.0, allowing it to use any of the implementations presented here.

We have packaged these implementations behind equivalent C++ and Python APIs in `libbdsg`. This software library will reduce the need for individual research groups to continually reimplement these core data structures and ease the development of algorithms that manipulate large, complex pangenome graphs. Moreover, the reduction in memory requirements makes it possible to move workloads that would otherwise need specialized high-memory machines

onto cheaper ones that often also have more processing power (for example, from Amazon's r4 instances to c5 instances). Combined with improvements in access speed over the previous VG dynamic graph implementation, substantial cost and time savings can be realized.

5.3 Implementation

5.3.1 Data model

Our libraries adopt node-labeled bidirected graphs as a formalism for sequence graphs. In a bidirected graph, nodes are considered to have left and right "sides", and edges connect two sides rather than two nodes. In bidirected sequence graphs, a node's sides correspond to the 5' and 3' ends of its DNA sequence. Nodes can be traversed either from left to right, which is interpreted as the forward strand of the sequence, or from right to left, which is interpreted as the reverse complement. This provides a natural means to encode DNA strandedness.

Longer sequences can be formed by concatenating the sequences of multiple adjacent nodes together. These nodes form a path, which is defined as a list of oriented nodes (either forward or reverse), such that the graph contains an edge between the adjacent sides of each pair of subsequent oriented nodes in the list[1]. Some paths correspond to sequences of interest, such as reference genomes or annotations of the reference. Because paths like these are so frequently important in practice, our graph formalism also includes a set of paths along with the graph's node and edges.

[1] Unlike the usage in many graph theoretic contexts, we do not intend the term path to indicate that these nodes must be distinct.

5.3.2 The HANDLEGRAPH interface

The `libhandlegraph` library describes an interface that exposes basic operations on our sequence graph data model. The HANDLEGRAPH model focuses on five fundamental entities in bidirected sequence graphs (Figure 5.1):

- *Nodes* identify pairs of complementary DNA strands and have unique numerical identifiers (IDs).
- *Strands* represent one strand of a node's DNA sequence.
- *Edges* link pairs of strands, in order.
- *Paths* represent sequences of interest as paths through the graph.
- *Steps* describe paths' visits to nodes' strands.

The defining feature of the model is that none of these entities are accessed directly. Instead, they are accessed via *handles*, which are references modeled after the concept of file handles. The handles are implemented as a data type with no methods and no prespecified meaning for its contents. Thus, we say that handles are "opaque" in that user code cannot usefully look inside them or manipulate their contents. Instead, the `libhandlegraph` interface requires the sequence graph implementation to provide queries that consume and produce handles, to expose graph information to users.

For example, we could obtain a handle to a strand from a HANDLEGRAPH implementation by providing a node's ID and an orientation (forward or reverse). We could then provide this handle to another of the graph's methods to obtain handles to this strand's neighbors, and

a further method would map the neighbors' handles to their node IDs. Alternatively, we could obtain a handle to a path from its name (e.g. "chr22"), and then iterate over handles to the path's steps to follow its course through the graph.

One benefit of this design is that any algorithm designed for one HANDLEGRAPH implementation can be applied to all other implementations. Since the actual contents of a handle are unspecified, this benefit is achieved while simultaneously maintaining flexibility in the implementation. Another benefit is that, since the user works only through handles that they cannot forge or modify, their ability to make mistakes can be restricted. For example, the interface can enforce the constraints that define valid paths through bidirected graphs during edge traversal. Furthermore, implementations can be made memory-safe by eliminating raw pointers and other direct access to graph elements.

5.3.3 Graph implementations

We consider five implementations of the HANDLEGRAPH model. To ground our experimental results, here we we provide a high-level overview of each implementation. Two implementations, VG and XG, have been described previously [70, 74]. The others are combined in the `libbdsg` library (`https://github.com/vgteam/libbdsg`), which provides three concrete implementations: HASHGRAPH, ODGI, and PACKEDGRAPH. Each implementation represents a different tradeoff in terms of speed, memory use, and capabilities. All of the implementations except XG are dynamic. They support efficient addition and deletion of nodes, edges, paths, and steps, as well as some specialized methods such as splitting a node into multiple shorter nodes. Table 5.1 provides a high level summary of the differences between the

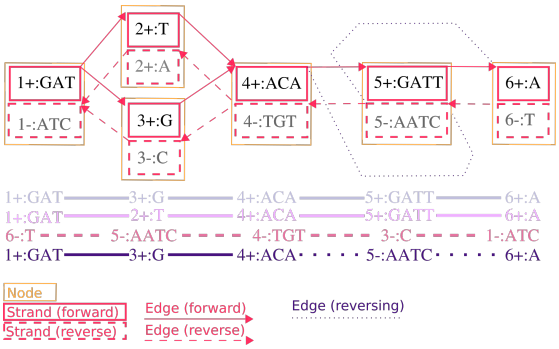


Figure 5.1: Entities in the bidirected sequence graph. Top: a variation graph showing *nodes* (yellow rectangles), each of which contain a forward and reverse *strand* (red solid and dashed rectangles, respectively). Strands show the node identifier, the direction ($+$ or $-$), and the sequence of the strand. Note that reverse strands show the reverse complement sequence of the forward strand. All *edges* are shown as connections between nodes, with forward-to-forward edges denoted by solid lines, and reverse-to-reverse edges denoted by dashed lines. Two edges that invert from forward to reverse and reverse to forward are shown with dotted lines. Edges run from the strand at their beginning to that at their end, as indicated by the arrowhead. Bottom: an illustration of four *paths*. Each has a name, and can be referenced by a handle, which are omitted for brevity. Each path is shown in its natural direction as a series of connected *steps* that refer to strands in the graph. The first two paths differ by a SNP, with one passing through `2+:T`, and the other through `3+:G`. The third path is the reverse complement of the first. The fourth is the same as the first, but contains an inversion, passing through `5-:AATC` rather than `5+:GATT`.

`libbdsg` implementations.

5.3.3.1 VG

We have extended the graph representation in VG, previously described in [74], to match the HANDLEGRAPH API. The backing data structures used remain the same. The graph entities are stored as objects in a backing vector, and referred to internally by hash tables that map between node identifiers and pointers into this vector. Edges are indexed in a hash table mapping pairs of handles to edge objects. Paths are stored in a set of linked lists, with a hash table mapping between nodes and path steps. This arrangement was tenable for the early development of algorithms working on variation graphs. Its inefficiency, caused by unnecessary overheads and data duplication, has resulted in significant difficulties for groups working with VG. The other HANDLEGRAPH implementations respond to the limitations of this approach. In version 1.22.0, vg was updated to use HASHGRAPH (below) as the default format, though it remains compatible with all implementations described in this paper via the HANDLEGRAPH API.

5.3.3.2 XG

XG was initially developed in response to the memory and runtime costs of VG, which prevent its application to large graphs. It additionally provides positional indexes over paths that are required for read mapping and variant calling, and is the graph data model used in most established bioinformatic operations on variation graphs [74, 86]. Unlike other HANDLEGRAPH implementations, XG is a static graph index. This permits a more powerful set of

efficient queries against the graph, especially for paths. The encoding is designed to balance speed and low memory usage. The topology of the graph is encoded in a single vector of bit-compressed integers, which promotes cache efficiency. Rank and select operations on succinct bit vectors are used to provide random access over the variable-length records, which each encode a node's sequence, ID, and edges. Embedded paths are encoded in variable-length integer vectors with Elias gamma encoding. Rank and select operations on succinct bit vectors also provide queries by base-pair position along paths. A detailed description of XG can be found in [70].

5.3.3.3 HASHGRAPH

HASHGRAPH is a relatively simple encoding, which is largely similar to the original VG graph. As such, it can be seen as a streamlined point of comparison for the other new dynamic graph implementations. However, the simplicity of this encoding has the benefit of allowing fast queries. Thus, even though HASHGRAPH still has relatively high memory requirements, it can still be useful in high memory compute environments or for small sequence graphs (such as subgraphs of genome graphs).

Like VG, HASHGRAPH encodes the topology of the graph in a hash table indexed by node IDs. However, what were separate hash tables in VG have been consolidated to avoid storing the keys multiple times. The hash table it uses is a drop-in replacement for the equivalent standard library (STL) data structure, and has been shown to outperform it in empirical evaluations [24]. Each hash table entry contains the sequence, an adjacency list of the edges in two STL vectors, and a vector indicating the path steps that the node can be found on. The

graph's paths are represented using doubly-linked lists to support efficient modification at any position.

In contrast to the more memory-efficient implementations, all of these data structures support computation in their native in-memory representation. Thus, the run time to access graph elements does not also include decompressing the data. This is how HASHGRAPH maintains its comparative speed advantage.

5.3.3.4 ODGI

ODGI (Optimized Dynamic Graph Implementation) is based on a node-centric encoding that is designed to improve cache efficiency when traversing or modifying the graph. This encoding is split between graph topology and paths, which is important for achieving a balance of runtime performance and memory usage on graphs with large path sets. It uses delta encoding of edges and path steps to reduce the cost of representing graphs with local partial order and sparsity, both of which are common features of pangenome graphs. ODGI is the default data model of the ODGI toolkit (`https://github.com/vgteam/odgi`), which provides high-level algorithms for graph manipulation and interrogation that are designed to work at the scale of large pangenomes.

In ODGI, each node $\mathcal{N} = (\mathcal{B}, \mathcal{P})$ is represented by a structure that contains a byte array $\mathcal{B} = (\mathcal{Q}, \mathcal{E})$ encoding its sequence and associated edges, and a compressed integer vector $\mathcal{P} = \mathcal{S}_1 \ldots \mathcal{S}_s$ describing the path steps that traverse it. The full graph model is simply an array of these node records $\mathcal{G} = \mathcal{N}_1 \ldots \mathcal{N}_{|\mathcal{G}|}$ with some additional data structures to allow for random access of paths by name, and to maintain important statistics about the size of the graph, its

node ID space, and its path set.

Each node's sequence Q is stored using a full byte per character at the start of the byte array $\mathcal{B}$. This allows ODGI to represent protein as well as DNA sequence graphs, and allows for copy-free reference to the node sequences. The edges that begin or end at the node are recorded in the remainder of $\mathcal{B}$, encoded as deltas between the rank of the other end of the edge and the current node.

ODGI stores paths as bidirectional linked lists that allow efficient insertions, deletions, and replacements of path steps. These paths are encoded in a manner that exploits common properties of pangenome graphs, and node-level data structures are organized to support efficient operation on graphs with very deep path coverage. The path steps $\mathcal{P} = \mathcal{S}_1 \dots \mathcal{S}_s$ on each node are recorded as a series of records in a dynamic integer vector which is compressed so that only the largest integer entry is stored at full bit-width [162]. Each step $\mathcal{S} = (p_{id}, \delta_p, \delta_n, r_p, r_n)$ contains a path identifier p_{id}, references to the previous δ_p and next δ_n node ID and strands on the path encoded as deltas relative to the current node, and the ranks of the previous r_p and next r_n steps among the path steps on their respective nodes. This path encoding scheme is similar to that used in the dynamic GBWT [190], but differs in that the paths are not prefix-sorted.

5.3.3.5 PACKEDGRAPH

PACKEDGRAPH is designed to have a very low memory footprint. The backing data structures are implemented using bit-compressed integer vectors. The bit-width of these vectors is chosen dynamically, starting with a bit-width of 1 and then reallocating the vector at a higher width whenever an edit operation introduces an integer that is too large to be represented with

the current width. In the typical case that the value of i-th entry in the vector is $O(i)$, these reallocations have an $O(1)$ amortized run time per edit.

Many of the integer vectors tend to also have entries that are highly correlated with their neighbors. PACKEDGRAPH exploits this characteristic to achieves greater compression by only storing one entry per fixed-size window at full bit-width. The rest of the entries are stored in a separate integer vector and expressed as a difference from that entry. Since the differences within a window tend to be small, this encoding keeps the bit-width for each window small as well.

The data associated with each node is recorded in several compressed integer vectors (at the same index in each). Contrast XG and ODGI, which encode data in a single vector to improve cache efficiency. Recording only one homogenous data type in a vector increases the correlation between neighboring values, which in turn improves compression. The adjacency list for the graph, the steps that each node is found on, and the paths themselves are represented using linked lists. The linked lists reside within the same bit-compressed integer vectors, where pointers are created by treating some integer entries as indexes into the vector itself. This pointer encoding also guarantees the technical condition that accessing the i-th entry is $O(i)$. The linked lists that occur on every node (the adjacency lists and node step lists) are included in a single vector across all nodes. This serves two purposes. First, the windowed compression scheme in the integer vectors is inefficient if lists are smaller than the window size, as is often the case. Second, due to the local partial order that is found in many pangenome graphs, neighboring nodes often connect to the same nodes and are found on the same paths as each other. Thus, the values they store are also highly correlated.

Model	HashGraph	ODGI	PackedGraph
Design goal	Simplicity, speed	Memory efficiency	Balanced speed/memory
Topology data structure	Hash table	Single integer vector	Several integer vectors
Topology compression	None	Delta encoding	Windowed bit compression
Sequence compression	None	None	Bit compression
Pointer encoding	Memory addresses	Delta-encoded ranks	Vector indexes

Table 5.1: Comparison of features between libbdsg graph implementations. The three graph implementations all use adjacency lists to encode graph topology and linked lists to encode paths. The differences in encoding these structures reflects different design goals for each implementation.

5.3.4 Python binding

We have implemented a Python binding to the graph implementations in `libbdsg` using Pybind11 [95]. This allows the data structures to be used in Python applications, significantly lowering the barrier-to-entry for pangenomic application developers. This functionality is documented at `https://bdsg.readthedocs.io`, including a tutorial. This documentation also serves as useful introduction to the HANDLEGRAPH API.

5.3.5 Code availability

Both `libhandlegraph` and `libbdsg` are open source under an MIT License. They are available on GitHub at `https://github.com/vgteam/libhandlegraph` and `https://github.com/vgteam/libbdsg`. Documentation for the two libraries, including the C++ handle graph API, HASHGRAPH, ODGI, and PACKEDGRAPH, is available at `https://bdsg`.

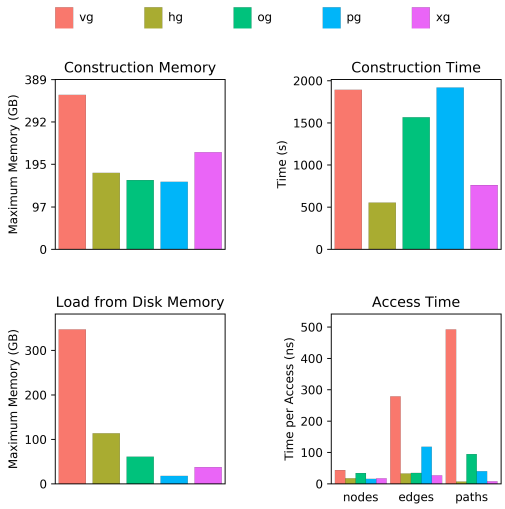


Figure 5.2: Performance on a graph of structural variants from the HGSVC. Abbreviations used here and in subsequent figures and tables: vg = VG, hg = HASHGRAPH, og = ODGI, pg = PACKEDGRAPH, xg = XG. All four new graph implementations compare favorably to VG. PACKEDGRAPH tends to be the most memory efficient, HASHGRAPH tends to be the fastest, and ODGI is balanced in between. XG provides good performance on both memory usage and speed, but it is static.

`readthedocs.io` alongside the documentation for the Python binding.

5.4 Evaluation

5.4.1 Human genome with structural variants

We measured the core operation performance of the four graph implementations and the graph class from the popular VG software (as implemented prior to version 1.22.0). In

particular, we measured 1) memory usage to construct a graph, 2) time to construct a graph, 3) memory usage to load an already-constructed graph, and 4) time to access nodes, edges, and steps of a path. These access operations are one of the major drivers of run time in pangenomic applications, such as VG's read mapping algorithm. Accesses were performed with a single thread, and the reported access time is the average time taken when accessing each graph element sequentially. All evaluations were performed on a 3.1 GHz Intel Xeon Platinum 8000 series processor. The presented results are from a graph describing the structural variants of the Human Genome Structural Variation Consortium [31], which was recently used to genotype structural variants [86]. Specifically, the graph consists of the GRCh38 primary scaffolds and 72,485 indel variants ranging in size from 50 bp to 76 kbp. The results generally match our expectations based on the implementations' design goals (Figure 5.2).

5.4.2 Genome graph collection

To compare the methods' performances across a wide variety of different graphs, we applied each to a collection of 2299 graphs collected during our research on graphical pangenomics. For each graph and graph implementation, we measured the same metrics described in the previous section as well as various graph properties including size, edge count, cyclicity, and path depth. We summarize these results in Figures 5.3 and 5.4.

For graph construction and loading, we observe similar trends as for the HGSVC graph. VG's performance in terms of memory usage is very poor, both during construction and load. For construction and load, all models exhibit largely linear scaling characteristics, outside of very small graphs where static memory overheads dominate. PACKEDGRAPH yields

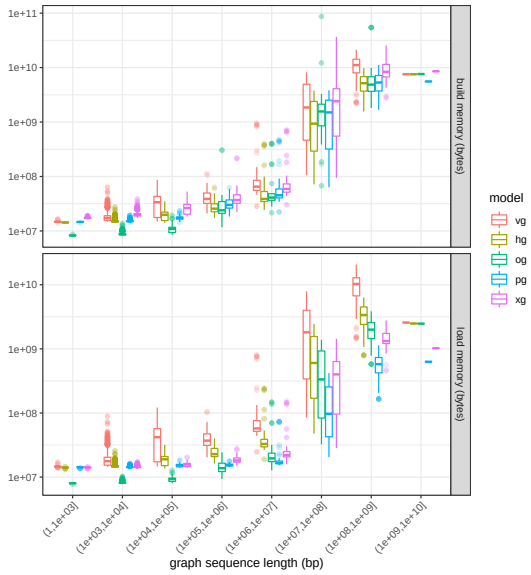


Figure 5.3: Memory requirements for model construction and loading. Memory costs versus graph sequence size for the graph collection, colored by HANDLEGRAPH model. The memory requirements for graph construction tend to be higher than those for loading the graph model. All methods show fixed overheads of several megabytes, seen in the flat tail to the left of both plots. Outside of this region, all methods show roughly linear scaling in both build and load costs per input base pair. The relative differences in memory costs appear to be stable between different methods across many orders of magnitude in graph size.

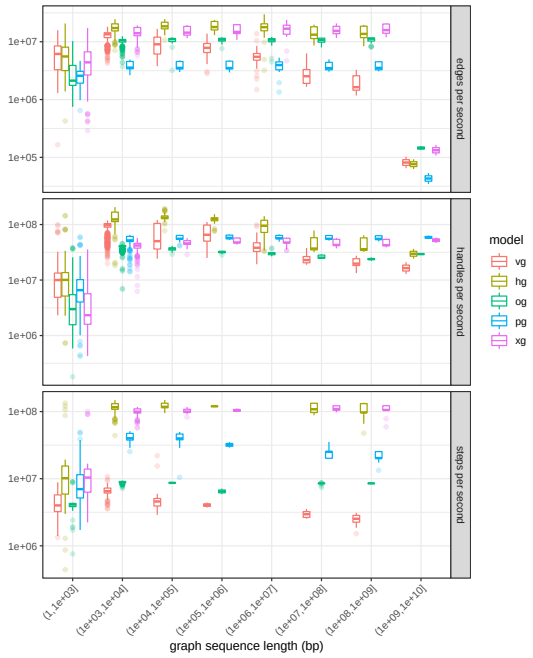


Figure 5.4: Graph element enumeration performance. Iteration performance for edges, nodes, and path steps for the full graph collection, shown in terms of elements per second. HASHGRAPH provides the highest performance for all element iteration types on smaller graphs, but this performance falls of with larger graphs, presumably due to scaling properties of the backing hash tables. The same pattern can be seen for VG, although the overall performance is worse. Although it has the worst edge iteration performance, PACKEDGRAPH provides good performance on node and path step iteration. The relative path encoding in ODGI yields poor performance on path iteration, and node decoding overheads appear to reduce its node iteration performance, but it has good graph topology traversal performance, perhaps due to cache efficiency of the edge encoding. XG provides excellent iteration performance in all cases.

the best memory performance for larger graphs (which are mostly the chromosomes of the 1000 Genomes Project graph), while for the medium-sized graphs in the collection ($\sim$1 Mbp), ODGI requires less memory.

For graph queries and iteration, the relative performance of the models is largely maintained across the entire range of graph sizes. However, we observe that the hash-based models (VG and HASHGRAPH) have very good performance for smaller graphs (in handle and edge enumeration) but decrease in throughput as the graph size increases. Smaller, less dramatic decreases in performance can be seen for the other implementations. For path enumeration, the highest-performing methods are XG and HASHGRAPH at approximately 10 times faster than ODGI, whose relative path storage is costly to traverse.

5.4.3 1000 Genome Project chromosome graphs

Variation graphs built from the 1000 Genomes Project (1000GP) variant catalog and the human reference genome have fairly homogenous and regular features. In addition, they have connected components of very different sizes, each corresponding to a chromosome. This provides a natural, fairly controlled means to explore the scaling behavior of our data structures. Moreover, graphs of this form are seeing increasing use in variant-aware resequencing analyses [47]. Thus, the performance of data structures on these graphs is of general interest.

We first evaluated the scaling performance of the various HANDLEGRAPH implementations relative to node count for each of the nuclear chromosomes in the 1000GP (Figure 5.5). We find that for all methods, load memory scales almost perfectly with node count, with an average $R^2 = 0.998$. Due to differences in variant density among the chromosomes, the average

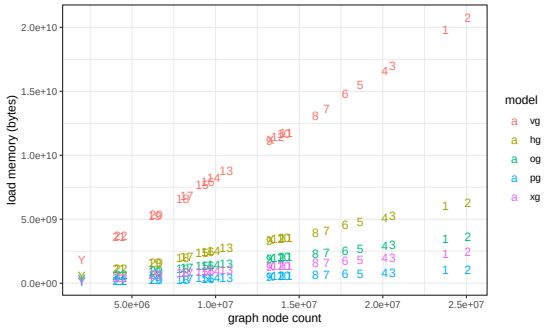


Figure 5.5: Load memory versus node count for chromosome graphs built from 1000 Genomes Project variants and GRCh37. For each method, memory requirements are more strongly correlated with the number of nodes in the graph ($R^2 = 0.998$) than with the graph sequence length ($R^2 = 0.986$). Although the memory requirements are dominated by graph sequence size, node count will increase with variant density. Methods generally incur an overhead for each node that is larger than the sequence length. Linear scales clarify that the absolute difference in performance between VG and the other methods is substantial.

Model	Build B/bp	Load B/bp	Iteration rate (millions) Node/s	Edge/s	Step/s
vg	80.2	77.2	24.6	2.8	2.9
hg	36.7	23.9	59.5	18.9	**127.2**
og	**30.3**	13.7	24.1	11.5	8.2
pg	37.6	**3.80**	**63.7**	4.6	24.3
xg	54.3	9.31	54.2	**20.5**	117.0

Table 5.2: Performance on 1000 Genomes Project chromosome graphs. Average build memory, load memory, and iteration times for graph elements for the chromosome-level graphs built from all the variants in the 1000 Genomes Project and the GRCh37 reference genome against which the variant set was originally reported. VG requires $\sim$ 20 times as much memory to load the graphs as PACKEDGRAPH, while even the most costly `libbdsg` model (HASHGRAPH) requires $\sim$ 1/3 as much memory. In these graphs, ODGI provides the lowest performance for handle iteration. However, in all other metrics, VG performs much worse than the other models.

correlation relative to sequence length is lower ($R^2 = 0.986$).

In Table 5.2, we report the average memory performance of the methods relative to graph sequence length, and also the iteration performance in terms of elements per second. We find that the best-performing method in terms of memory usage is PACKEDGRAPH, which consumes around $1/20$th the memory of VG per base-pair of graph in the 1000GP set. Moreover, it provides much better iteration performance for nodes (handles), edges, and path steps. HASHGRAPH and XG have similar iteration performance, but XG, by virtue of its use of compressed, static data structures, requires less than half as much memory. ODGI optimized for efficient dynamic operations on graphs with higher path coverage, and in general is not as performant as other methods on this set.

5.5 Discussion

We have presented a set of simple formalisms, the HANDLEGRAPH abstraction, which provides a coherent interface to address and manipulate the components of a genome variation graph. To explore the utility of this model, we implemented data structures to encode variation graphs and matched them to this interface. This allowed us to directly compare these HANDLEGRAPH implementations on a diverse set of genome graphs obtained during our research. These experiments reveal that genome graphs need not pay the computational expense of the early versions of VG. The best-performing models require an order of magnitude less memory than VG while providing higher performance for basic graph access operation and element iteration. For these reasons, VG has transitioned to using these newer graph implementations.

The efficiency of these methods and their encapsulation within a coherent programming interface will support their reuse within a diverse set of application domains. Variation graphs have deep similarity with graphs used in assembly; these libraries could be used as the basis for assembly methods. They could also be used for genotyping and haplotype inference methods based on graphs [69].

Ongoing work is establishing large numbers of highly-contiguous whole genome assemblies for humans (`https://humanpangenome.org/`). Improvements in sequencing technology are likely to make such surveys routine. It is natural to consider a pangenome reference system, based on the whole genome alignments of such assemblies, as the output of these pangenome projects. Recent results demonstrate that many basic bioinformatic problems can be generalized to operate on such structures. Should these pangenome representations become

common or standard, then variation graph data structures like those we have presented here will form the basis for a wide range of pangenomic methods.

Chapter 6

Automated index coordination within the VG toolkit

6.1 Introduction

Since its initial publication, VG [74] has become one of the most widely used software tools for graph-based pangenomics. Although the VG toolkit has many functionalities, it is best known for read mapping to sequence graphs. Indeed, VG contains three separate mapping algorithms:

1. vg map: the original, highly accurate mapping algorithm [74]
2. vg giraffe: the much faster and still accurate haplotype-based mapping algorithm [191]
3. vg mpmap: the splice-aware RNA-seq mapping algorithm that can produce multipath alignments [187]

Each of these mapping tools depends on a sizeable body of research in specialized data structures and algorithms [32, 62, 189, 190]. As a result, they each require a different set of indexes to be built before mapping. Historically, navigating the indexing process has been a pain point for VG's users. The indexing algorithms and their documentation are spread across several VG subcommands, and the steps need to be applied in a specific order to produce valid, usable results.

The VG autoindex utility is designed to alleviate the pain involved in this process. Rather than having an interface based on which index the user wants to produce, it has an interface based on which mapping tool they want to run. The inputs are all common interchange formats like FASTA, VCF, and GFA. Internally, autoindex has the logic of the VG team's best practice indexing pipelines built in. Power users might still find need for the individual indexing subcommands, but the goal of autoindex is to produce indexes for any common use case in a single, easily-understood shell command.

6.2 Methods and implementation

The central abstraction in autoindex is the *recipe graph* (Figure 6.1). This graph is a bipartite directed acyclic graph with two classes of nodes: *file nodes* and *recipe nodes*. The file nodes correspond to either input data (provided in interchange formats) or to VG indexes. The recipe nodes correspond to algorithms to produce VG indexes from a set of file nodes, which may correspond either to input data or to intermediate indexes. Each recipe node expresses a many-to-many relationship between file nodes: the recipe may require multiple files

as input, and it may produce multiple files as output. This graph is hard-coded into VG. Each of the constituent recipes is based on the best practices developed for the VG developers' own pipelines.

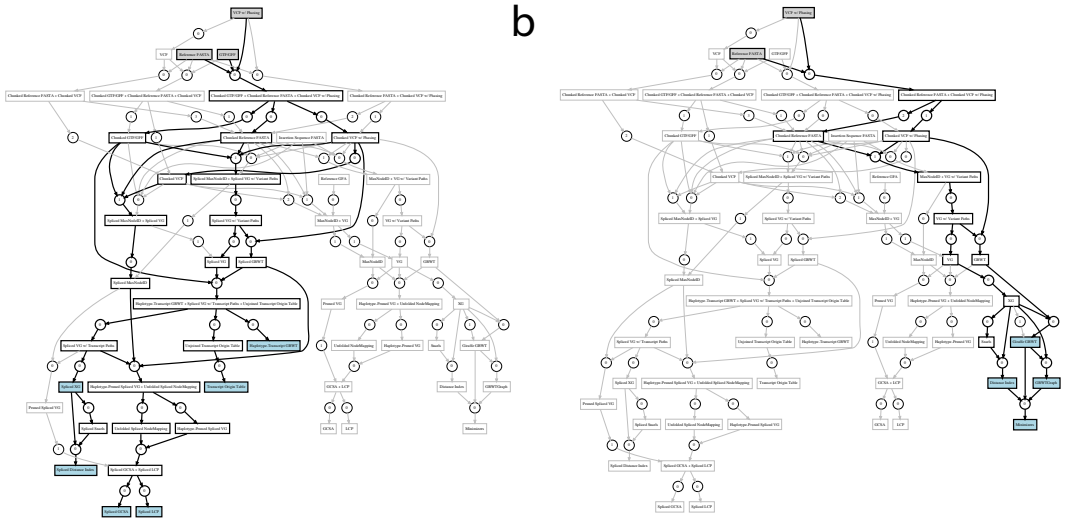


Figure 6.1: Two plans in the recipe graph. Plans for the indexes required by **a** `vg mpmap` and `rpvg` and by **b** `vg giraffe` are highlighted in the recipe graph. Rectangular nodes correspond to indexes or data files, and circular nodes correspond to recipes. Lower numbers on the recipe nodes indicates higher priority. Gray shading indicates provided data, and blue shading indicates the target indexes being constructed.

6.2.1 Planning index construction pipelines

The autoindex utility is equipped with an algorithm that determines the best pipeline, referred to as a *plan*, to construct any index in the recipe graph using a given set of inputs. Recipes are labeled with a priority level: a total ordering that is used to determine which recipe

is preferred when more than one recipe could be used to construct the same index. A plan is considered suboptimal whenever a higher priority recipe could be used to construct any of its indexes. In general, higher priority recipes correspond to improved versions of indexes that can be constructed using additional information. For example, path queries can be improved if the variants in a VCF file are phased [190].

The planning algorithm uses an exhaustive search to find the optimal plan. To begin, all file nodes that have been provided as input data are labeled complete. The search begins by initializing a queue with the target index. In each iteration, the lowest-valued file node (according to the partial order defined by the recipe graph, which is a DAG) on the queue is de-queued. If this file node is labeled complete, nothing is added to the queue. Otherwise, the file node's highest-priority recipe is added to the plan and the inputs for that recipe are added to the queue. If there is no recipe to construct the index, the search backtracks through the plan marking recipes impossible until finding a file node that has remaining recipes that have not yet been marked impossible. This algorithm is exponential in the worst case. However, the recipe graph is fairly small, and in practice the planning process takes a negligible amount of time.

The indexes required for each of VG's mapping tools are hard-coded into the autoindex utility. This is what allows autoindex to provide its tool-oriented interface. The user requests whatever indexes are required to use a tool. The autoindex utility then looks up a list of indexes that are sufficient to run that tools and finds the optimal plan to construct each of them based on the available data. The individual index's plans are then merged into one plan, and the pipeline is executed.

This design has several benefits. If pipelines were described explicitly, it would be

challenging to design them to adapt appropriately when different data sources are available. In addition, many intermediate indexes are shared across different pipelines. Constructing these indexes in explicit plans would require either significant duplicated code or complicated hard-coded logic. Instead, the autoindex utility pushes all of that complexity into the planning algorithm. Each recipe can be developed as an atomic unit, which makes them easy to maintain. It is also easy to extend the indexing ecosystem within VG, which can be accomplished by adding file nodes and recipes. The logic of integrating these new indexes into pipelines is then determined automatically.

6.2.2 Improving the computational performance of indexing pipelines

The VG autoindex utility is designed to perform well on a moderately large compute server, which is generally the computational environment that pangenomics practitioners operate in. Accordingly, there is likely to be a significant number of compute threads available, as well as a reasonably large bank of RAM. In our experience, it is essential that indexing pipelines utilize upwards of 8 threads in order to be practical at the scale of eukaryotic pangenomes. Accordingly, autoindex must be designed so that it can use these compute resources effectively.

Autoindex's primary strategy for multithreading is to chunk input data according to chromosomes or contigs. Many indexes can be constructed fully in parallel across these chunks. However, in the final steps, it is generally necessary to merge these chunks with a single-threaded algorithm to produce a usable index for mapping. Up until that point, a consistent chunking of contigs is maintained across all recipes. The autoindex utility also uses a scheduling algorithm that attempts to stay within the constraints of available memory. It estimates the

memory required to construct each chunk, and only executes the construction algorithm if there is sufficient memory available.

Part IV

Graph theoretic contributions

Chapter 7

Identifying hierarchical sites of variation in a pangenome graph

7.1 Preamble

This chapter consists of the majority of the paper "Superbubbles, Ultrabubbles and Cacti", which was presented at the 2017 meeting of RECOMB and subsequently published in *Journal of Computational Biology* [154]. I was not a primary author on this publication. However, I contributed in whole the sections titled *Compatible Snarl Families* and *Ultrabubbles and Cacti*, including both the mathematical proofs and the actual writing. This chapter includes most of the rest of the main text, as it provides motivation for the problem and introduces mathematical notation that is used in my section. These portions were primarily written by Benedict Paten. I have omitted the results section, which is not necessary to understand my contributions. I have also omitted proofs in the appendices for all theorems that I did not prove

myself.

7.2 Introduction

Graphs are used extensively in biological sequence analysis, where they are often used to represent uncertainty about, or ensembles of, potential nucleotide sequences. Several subtypes have become especially prominent for sequence representation, in particular the De Bruijn graph [51, 159], the string graph [143], the breakpoint graph [4, 158] and the bidirected graph (aka sequence graph) [60, 135].

In the context of de novo sequence assembly several characteristic types of subgraph are recognised, in particular the *bubble* [218], a pair of paths that start and end at common source and sink nodes but are otherwise disjoint. In the context of sequence analysis, a bubble can represent a potential sequencing error or a genetic variation within a set of homologous molecules. An efficient algorithm for bubble detection was proposed by [20].

A generalization of the notion of a bubble, the superbubble is a more complex subgraph type in which a set of (not necessarily disjoint) paths start and end at common source and sink nodes. This problem was initially proposed by [150], who gave a quadratic solution. [22] recently provided a linear time algorithm for superbubbles on directed acyclic graphs (DAGs). This result, when paired with a previous linear time transformation of the problem of superbubbles on directed graphs to superbubbles on DAGS [201], yields a linear cost solution for computing superbubbles on digraphs. For a review of superbubbles and their use in sequence analysis see [88]. In this paper we generalize the idea of superbubble to the more general case of

a bidirected graph, connect a slight generalization of the superbubble, which we call the ultra-bubble, and show how it relates to the decomposition of the graph into 2- and 3-edge connected components.

7.3 Methods

7.3.1 Directed, Bidirected and Biedged Graphs

A *bidirected graph* $D = (V_D, E_D)$ is a graph in which each endpoint of every edge has an independent orientation (denoted either "left" or "right"), indicating if the endpoint is incident with the left or right *side* of the given vertex. The sides of D are therefore the set $V_D \times \{left, right\}$, and each edge in E_D is a pair set of two sides (Fig. 7.1). We say for all $x \in V_D$, $(x, left)$ and $(x, right)$ are *opposite sides*.

Any digraph is a special case of a bidirected graph in which each edge connects a left and a right side (by convention we here consider the right side to be the outgoing side and the left side the incoming side, so that the conversion from a digraph to a bidirected graph is determined; see Fig. 7.1).

A *biedged graph* is a graph with two types of edges: *black edges* and *grey edges*, such that each vertex is incident with at most one black edge (Fig. 7.1(C)).

For any bidirected graph D there exists an equivalent biedged graph $B(D) = (V_{B(D)}, E_{B(D)})$ where:

- $V_{B(D)} = V_D \times \{left, right\}$, the sides of V_D.
- $E_{B(D)} = S_{B(D)} \cup E_D$, where E_D are the grey edges,

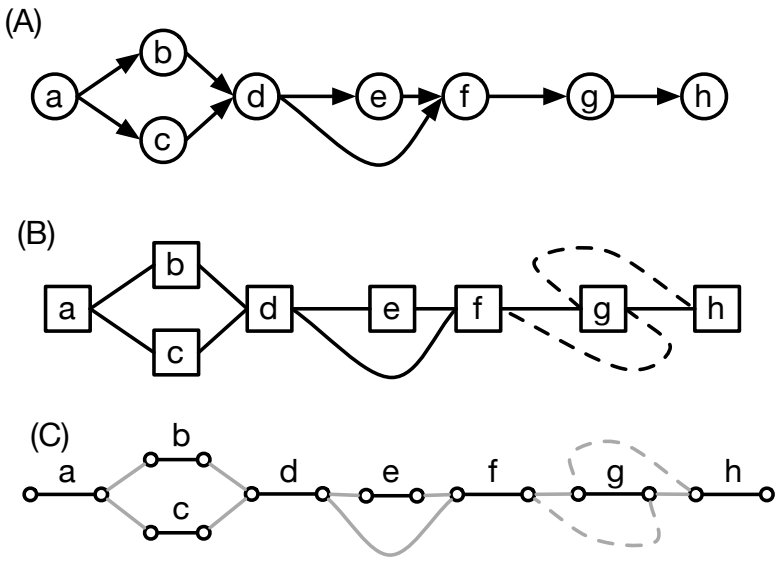


Figure 7.1: (A) A digraph. (B) A bidirected graph. Each node is drawn as a box and the orientation for each edge endpoint is indicated by the connection to either the left or right side of the node. The graph excluding the dotted edges is the equivalent bidirected graph for the digraph in (A); the dotted edges encode an inversion that cannot be expressed in the digraph representation. (C) A biedged graph equivalent to the bidirected graph shown in (B).

- and $S_{B(D)} = \{\{(x, left), (x, right)\} | x \in V_D\}$ are the black edges.

For a vertex $x \in V_{B(D)}$ we use the notation $\hat{x}$ to denote its opposite side.

Clearly the bidirected and biedged representations are essentially equivalent, and the choice to use either one is largely a stylistic consideration. For the remainder of this paper we will mostly use the biedged representation. As any digraph is a special case of a bidirected graph and any bidirected graph has an equivalent biedged graph, so any digraph has an equivalent biedged graph.

7.3.2 Directed Walks on Biedged and Bidirected Graphs

A directed walk on a bidirected graph is a walk that at each visited vertex exits the opposite side to that which it enters. On a biedged graph a directed walk is equivalent to a walk that alternates between black and grey edges. A directed cycle is a closed directed walk that starts and ends either on the same side (e.g. a self-loop edge), or on opposites sides of a vertex (in which case the start and end is arbitrary due to symmetry). A bidirected or biedged graph is acyclic if it contains no directed cycles.

These definitions are a generalization of a directed walk on a digraph. In a bidirected representation of a digraph all edges in a directed walk are all left-to-right or all right-to-left. A directed walk on a general bidirected (or biedged) graph can mix these two types and additionally include edges that do not alternate the orientation of their endpoints (e.g. left-right, right-right and left-left edges).

Given these generalizing relationships, clearly a digraph $\mathbf{D}$ is acyclic iff $B(\mathbf{D})$ is acyclic. Note that any acyclic biedged graph can also be converted into an equivalent directed acyclic graph (DAG):

Lemma 1 *For any acyclic biedged graph $B(D)$ there exists an isomorphic biedged graph $B(\boldsymbol{D})$ such that $\boldsymbol{D}$ is a DAG.*

Proof *For each connnected component in $B(D)$, use a depth first search (DFS) beginning at side x to label the sides either 'red' or 'white': If x is not already labelled then label x red and $\hat{x}$ white. For each grey edge incident with $\hat{x}$, if the connected side is not labeled, label the connected side red and continue recursively via DFS. In this way all the sides in the connected*

component containing x will be labeled in a single DFS. If during the recursion the connected side encountered is already labelled then it must be labeled red, else there would exist a directed cycle, a contradiction. Use the labelling to create $B(\boldsymbol{D})$, isomorphic to $B(D)$ but replacing the orientation of the sides so that each side labeled white is a left side and each side labeled red is a right side. All edges in $B(\boldsymbol{D})$ connect a left and a right side.

7.3.3 Superbubbles, Snarls and Ultrabubbles

Repeating the definition from [150], any pair of distinct vertices (x, y) in a digraph **D** is called a *superbubble* (Fig. 7.2(A)) if:

- *reachability*: y is reachable from x.
- *matching*: The set of vertices, X, reachable from x without passing through y is equal to the set of vertices from which y is reachable without passing through x (passing through here means to enter and then exit a vertex on the path).
- *acyclicity*: The subgraph induced by X is acyclic.
- *minimality*: No vertex in X other than y forms a pair with x that satisfies the criteria defined above, and similarly for y.

We call the subgraph induced by X the *superbubble subgraph*.

To generalize superbubbles for biedged graphs we introduce the notion of a snarl, a minimal subgraph in a biedged graph whose vertices are at most 2-black-edge-connected (2-BEC) to the remainder of the graph (two vertices in a biedged graph are k-black-edge-connected

(k-BEC) if it takes the deletion of at least k black edges to disconnect them). In a biedged graph $B(D)$ a pair set of distinct, non-opposite vertices $\{x,y\}$ are a *snarl* (Fig. 7.2(B)) if:

- *seperable*: The removal of the black edges incident with x and y disconnects the graph, creating a *separated component* X containing x and y and not $\hat{x}$ and $\hat{y}$.
- *minimality*: No pair of opposites $\{z,\hat{z}\}$ in X exists such that $\{x,z\}$ and $\{y,\hat{z}\}$ fulfils the above criteria.

We call a vertex not incident with a grey edge a *tip* [218]. In a biedged graph $B(D)$ a snarl is an *ultrabubble* if its separated component is acyclic and contains no tips.

The following shows that a superbubble in a digraph is an ultrabubble in the equivalent biedged graph.

Lemma 2 *For any superbubble (x,y) in a digraph $\boldsymbol{D}$, the pair set $\{x' = (x,right), y' = (y,left)\}$ is an ultrabubble in $B(\boldsymbol{D})$.*

Proof *Let d and e be the black edges incident with x' and y', respectively, and let X be the superbubble subgraph of (x,y).*

We start by proving that $\{x',y'\}$ satisfies the separable criteria. As y is reachable from x by definition there exists a directed path in $B(\boldsymbol{D})$ between x' (the right side of x) and y' (the left side of y) that excludes d and e. After the deletion of these black edges x' and y' therefore remain connected. If the separable criteria is not satisfied the deletion of d and e must therefore not disconnect x' and y' from either or both $\hat{x}'$ and $\hat{y}'$, without loss of generality assume x' (and therefore y') remains connected to $\hat{x}'$.

If $\hat{x'}$ is on a directed walk from x' that excludes d then the addition of d to this walk defines a directed cycle in $B(\boldsymbol{D})$. As all nodes reachable from x are in the separated component X, the existence of this cycle in $B(\boldsymbol{D})$ implies the existence of a corresponding directed cycle in X, a contradiction.

If there exists a non-directed walk from x' to $\hat{x'}$ then let z' be the last node on the walk from x' such that the subwalk between x' and z' is a directed walk. By definition, there exists directed walk from z' to y'. The next node on the walk from x' to $\hat{x'}$ after z' is, by definition, not reachable from x' but y' must be reachable from this node. This implies a contradiction of the matching criteria for the corresponding nodes in X.

We have therefore established that $\{x', y'\}$ fufills the seperable criteria. We have already established that iff a digraph is acyclic its equivalent biedged graph is acyclic, therefore the seperated component of $\{x', y'\}$ is acyclic. As every node in X is both reachable from x and on a path from y, the separated component clearly contains no tips.

It remains to prove that $\{x', y'\}$ fufills the minimality criteria. If $\{x', y'\}$ do not satisfy the minimality criteria without loss of generality there exists a node z' in the separated component of $\{x', y'\}$ such that $\{x', z'\}$ are separable. It follows that all directed paths from x' to y' that exclude d and e visit z', and for the node z in $\boldsymbol{D}$ contained in z', (x, z) fulfills (clearly) all the superbubble criteria, a contradiction.

7.3.4 Cactus Graphs

A cactus graph is a graph in which any two vertices are at most two-edge connected [84]. In a cactus graph each edge is part of at most one simple cycle, and therefore any two

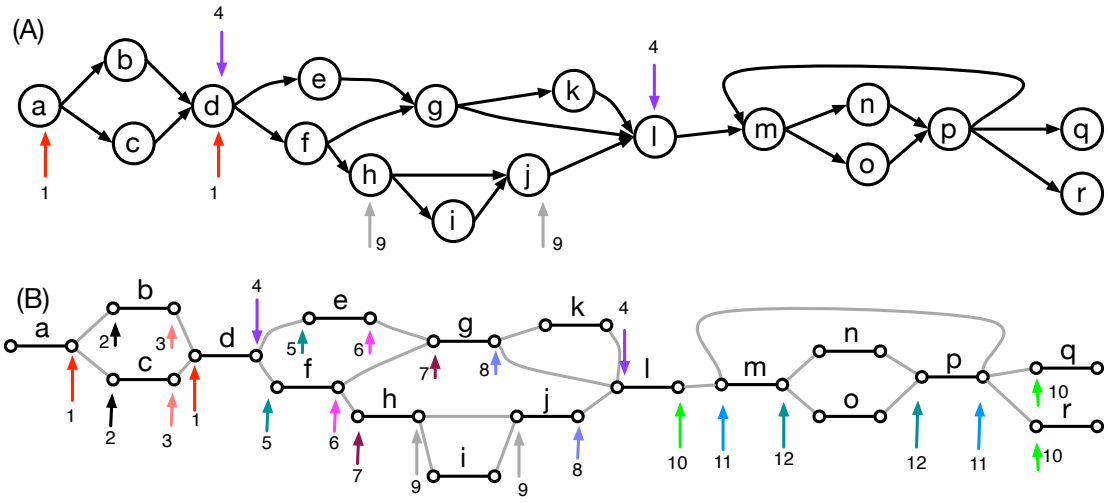


Figure 7.2: (A) Superbubbles in a digraph. The superbubbles are indicated by pairs of numbered arrows, numbered consistently with (B). (B) A biedged graph representation of the digraph in (A). The snarls are illustrated by numbered arrows, the ultrabubbles are those numbered 1, 4, 9 and 12. Note, a side incident with a black bridge edge may be in multiple snarls (see snarls numbered 10).

simple cycles intersect at most one vertex.

For a graph $G = (V_G, E_G)$ let $G' = (V_{G'}, E_{G'})$ be a multigraph created by *merging* subsets of the vertices, such that:

- $V_{G'}$ is a partition of V_G,
- $E_{G'} = \{\{a_{G'}(x), a_{G'}(y)\} | \{x,y\} \in E_G\}$ is a multiset.

where $a_{G'} : V_G \to V_{G'}$ is a graph homomorphism that maps each vertex in V_G to the set in $V_{G'}$ that contains it.

Merging all equivalence classes of 3-edge connected (3-EC) vertices in a graph results in a cactus graph [153].

For a biedged graph $B(D)$ let $C(D)$ be the cactus graph created by first contracting all the grey edges in $B(D)$ then for each equivalence class of 3-EC vertices in the resulting graph merging together the vertices within the equivalence class (Fig. 7.3(A-C)). As with G' and G, $V_{C(D)}$ is a partition of the vertices of $V_{B(D)}$, and $E_{C(D)} = \{\{a_{C(D)}(x), a_{C(D)}(y)\} | \{x,y\} \in E_{B(D)}\}$ is a multiset.

For a vertex $x \in V_{B(D)}$ we call $a_{C(D)}(x)$ its *projection* (in $C(D)$). Similarly for a set of vertices $X \subset V_{B(D)}$ we call $\{a_{C(D)}(x) | x \in X\}$ the projection of X (in $C(D)$). Let $b_{C(D)}(x) = \{a_{C(D)}(x), a_{C(D)}(\hat{x})\}$, which is the projection of the black edge incident with x in $C(D)$.

Appendix 1 of [154] (omitted in this chapter) gives lemmas that make explicit the relationship between the edge connectivity of vertices in $B(D)$ and $C(D)$, and which we use to prove the relationship between the snarls of $B(D)$ and $C(D)$.

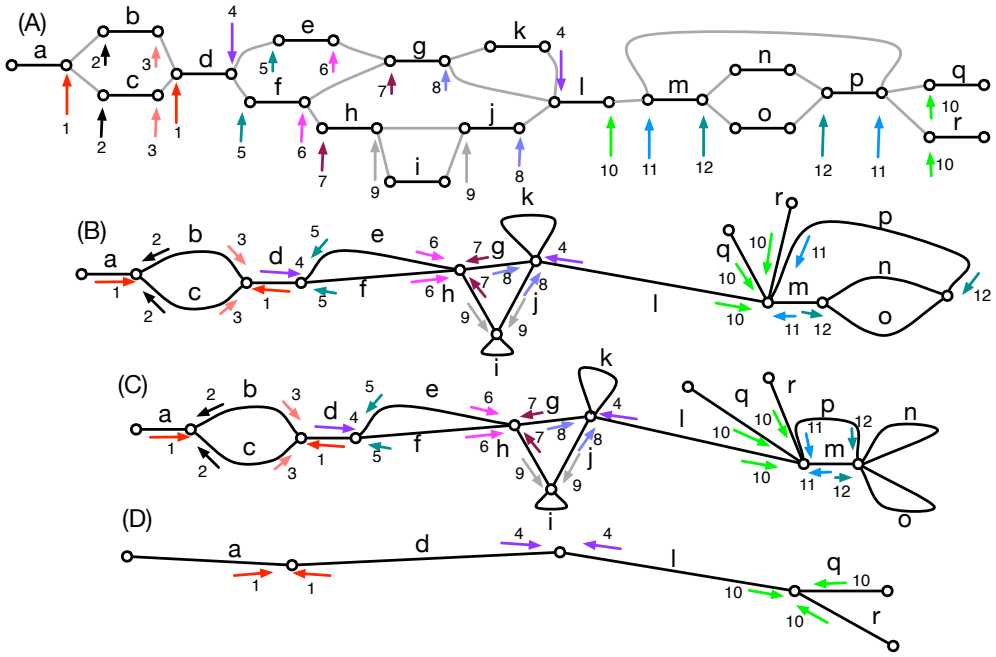


Figure 7.3: (A) A biedged graph $B(D)$ with the snarls indicated by pairs of numbered arrows. (B) The graph in (A) after contracting the grey edges. (C) The cactus graph $C(D)$ for $B(D)$, constructed by merging the vertices in each 3-EC in (B). (D) The bridge forest $D(D)$, constructed by constracting the edges in simple cycles in (C).

7.3.5 Snarls and Cacti

A pair set of distinct vertices $\{x,y\}$ in $B(D)$ are a *chain pair* if they project to the same vertex in $C(D)$ and their incident black edges project to the same simple cycle in $C(D)$ (e.g. pairs of arrows in simple cycles in Fig. 7.3(C)). A cyclic sequence of chain pairs within the same simple cycle in $C(D)$ and ordered according to the ordering of this simple cycle is a *(cyclic) chain*. Contiguous chain pairs in a chain share two opposite sides of a black edge in $B(D)$.

For a cactus graph $C(D)$, the graph $D(D)$ resulting from contracting all the edges in simple cycles in $C(D)$ is a called a *bridge forest* (Fig. 7.3(D)).

A pair set of distinct vertices $\{x,y\}$ in $B(D)$ are a *bridge pair* if they project to the same vertex in $D(D)$ and both their incident black edges are bridges (e.g. pairs of arrows numbered 1 and 2 in Fig. 7.3(D)). A maximum sequence of bridge pairs within $D(D)$ connected by incident nodes with degree two is an *(acyclic) chain*. As with chain pairs, contiguous bridge pairs in a chain share two opposite sides of a black (bridge) edge in $B(D)$.

Theorem 1 *The set of snarls in $B(D)$ is equal to the union of chain pairs and bridge pairs.*

Proof *Follows from Lemmas 10 and 11 given in Appendix 2 in [154] (omitted from this chapter).*

Given Theorem 1 to calculate the set of snarls for a given biedged graph it is sufficient to calculate the cactus graph to give the set of snarls that map to chain pairs and the bridge forest to calculate the set of snarls that map to bridge pairs. Constructing a cactus graph of the type described for a biedged graph is linear in the size of the biedged graph (using the algorithm

described in [153]), and clearly the cost of then calculating the bridge forest from the cactus graph is similarly linear. The number of chain pairs is clearly linear in the size of the biedged graph, however, the number of bridge pairs is potentially quadratic in the number of bridge pairs, so enumerating these latter snarls has potentially worst case quadratic cost in terms of the size of the biedged graph. Below we consider ways to prune the set of snarls by using their natural nesting relationships to create a hierarchy of snarls that is at most linear in the size of the biedged graph.

7.3.6 Compatible Snarl Families

One particularly attractive feature of superbubbles is that they have nested containment relationships. That is, superbubbles have subgraphs that are either strictly nested or disjoint. Accordingly, a digraph is partitioned into a set of top level superbubble subgraphs and other graph members not contained in a superbubble subgraph, and each top level superbubble component then contains one or more child superbubbles, forming a tree structure. The situation is more complex for snarls. The separated component of snarls can overlap (Fig. 7.4) such that each partially contains the other. To create a properly nested hierarchy of snarls it is therefore necessary to exclude some snarls.

We will call a family of snarls *compatible* if all pairs of distinct snarls in the family have snarl subgraphs that are either disjoint or nested. A compatible family of snarls has a nesting structure that is a forest, similar to superbubbles. The following theorem provides a sufficient condition for constructing such a family in many bidirected graphs.

Theorem 2 *In a connected biedged graph with at least one black bridge edge, the family of*

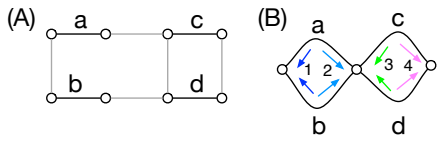


Figure 7.4: Overlapping snarls. (A) A bidirected graph, its corresponding (B) cactus graph. The snarl numbered 2 contains the snarl numbered 4, similarly the snarl numbered 3 contains the snarl numbered 1. The snarls numbered 2 and 3 overlap.

snarls whose subgraphs have no black bridge edges is compatible.

In addition, the next theorem shows that this family of snarls is a generalization of ultrabubbles.

Theorem 3 *No ultrabubble contains a black bridge edge in its subgraph.*

Proofs of these theorems are included in Appendix B.

The bridge edge condition can also be used to construct a compatible family of snarls in a graph with no black bridge edges. To do so, we break one black edge into two tips. Each of these tips is then a bridge edge, so the family of snarls we construct from the modified graph is compatible. However, the family of snarls we obtain will depend on our choice of a black edge to break. Heuristically, an edge corresponding to a highly conserved genomic element should be chosen, since by construction it will not occur in any snarl's subgraph.

Given a snarl decomposition, the following algorithm will filter them down to the compatible family we have described:

- Iterate over the black bridge edges of the graph (i.e. the edges of $D(D)$)
- For a bridge edge $(u, \hat{u})$, if either u or $\hat{u}$ is the boundary of a snarl, mark that snarl as not containing $(u, \hat{u})$.

- Initialize a queue with u and $\hat{u}$, and traverse outward in breadth-first order, ignoring restrictions on directed biedged walks.

- Upon reaching a node x that is a boundary for a snarl $\{x, y\}$, if neither y, $\hat{x}$, nor $\hat{y}$ have been traversed, mark the snarl as containing $(u, \hat{u})$.

- Upon reaching a node $\hat{x}$ whose opposite is boundary for a snarl $\{x, y\}$, if neither $\hat{y}$, x, nor y has been traversed, mark the snarl as not containing $(u, \hat{u})$.

- After completing every traversal, retain only snarls that were never marked as containing a black bridge edge.

The validity of this algorithm is proven by Lemma 19 in Appendix B. Naively, this algorithm requires $O(|E_{B(D)}|(|V_{B(D)}| + |E_{B(D)}|))$ time for the traversals, and $O(|V_{B(D)}|^2)$ to mark all snarls. However, we can implement optimizations that improve on this behavior. First, we can also stop the BFS traversals whenever they encounter a bridge edge. Lemmas 20 and 21 in Appendix B demonstrate that the portion of the BFS traversal after a bridge edge is redundant. This reduces the time required for the traversal to $O(M(|V_{B(D)}| + |E_{B(D)}|))$, where $M = \max_{v \in V_{D(D)}} \deg v$. In general, $M = O(|E_{B(D)}|)$, so this does not improve over the worst case asymptotic bound. However, in many practical cases M is approximately constant.

We can also reduce the total number of snarls we need to filter by neglecting to produce some snarls a priori. The quadratic bound on the number snarls is due to the fact that there is a bridge pair for all pairs of edges incident on a node in $D(D)$. However, Lemma 15 in Appendix B shows that none of these bridge pairs will pass the filter. Accordingly, we can reduce the set of snarls we consider to only chain pairs and bridge pairs that project to nodes of

degree 2 in $D(D)$, which we call *simple* bridge pairs. This reduces the total number of snarls to $O(|V_{B(D)}|)$.

7.3.7 Ultrabubbles and Cacti

Given Theorem 1, to determine the ultrabubbles in $B(D)$ it is sufficient to check for each chain and bridge pair if the separated component is acyclic and contains no tips.

Using Theorem 3 we can restrict the search to snarls whose separated component does not contain a black bridge edge. This implies that we need only consider bridge pairs whose projection in $D(D)$ is a node whose degree is two, we call such bridge pairs *simple*. The number of simple bridge pairs must be less than the cardinality of $D(D)$, and therefore the total number of chain pairs and simple bridge pairs is less than or equal to $|E_{B(D)}|$. Using $D(D)$ and $C(D)$, which both can be constructed in $O(|E_{B(D)}| + |V_{B(D)}|)$ time, we can clearly enumerate the set of simple chain pairs and bridge pairs in $O(|E_{B(D)}| + |V_{B(D)}|)$ time.

A simple algorithm to find the set of ultrabubbles enumerates all chain pairs and simple bridge pairs and checks for each the acyclicity and tipless requirement using a depth first search, and is therefore worst case $O((|E_{B(D)}| + |V_{B(D)}|)^2)$ time.

7.4 Discussion and Conclusion

We have presented a partial decomposition of a bidirected graph into a set of nested snarls and ultrabubbles. We believe this solves an important problem in using graphs for representing arbitrary genetic variations by defining a decomposition that determines sites and

alleles.

As the decomposition is only partial, not all elements in a graph will necessarily fit into one of the ultrabubbles. However, we demonstrate that for an existing large library of variation (1000 Genomes) the large majority of sites are either invariant or described by simple, top-level ultrabubbles.

For bases outside of these easy sites it is possible to imagine further subclassification. For example, classifying snarls that contain tips but are acyclic might define a useful class of subgraph common in some subproblems (e.g. sequence assembly). Some structures representing dense or overlapping collections of sequence polymorphisms, insertions and deletions cannot be fully described using nested ultrabubbles. We have previously shown that a generalization of the separability criterion for ultrabubbles can describe sites in these cases. [177]. Similarly, characteristic structures representing genomic phenomena, such as inversions and translocations, are imaginable. Beyond our initial investigation, a more thorough evaluation of how much of a graph fits within a snarl, ultrabubble, or one of these more complex structures would be a useful exercise. We propose that the compatible family of snarls we constructed provides one path forward in this endeavor.

We can also envision that the nesting structure of snarls could play a powerful role in decomposing genotyping problems. Nested graph structures often arise from nested indels and substitutions.

In the context of assembly, various error correction algorithms have been proposed to remove graph elements and reduce the complexity of the graph. This increases the fraction of the graph that is contained within an ultrabubble structure. We foresee the cactus graph structure

providing a useful basis for exploring such algorithms.

7.5 Acknowledgements

This work was supported by the National Human Genome Research Institute of the National Institutes of Health under Award Number 5U54HG007990 and grants from the W.M. Keck foundation and the Simons Foundation. This work benefitted from numerous conversations with David Haussler and Daniel Zerbino.

Chapter 8

Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs

8.1 Preamble

This chapter consists of the full text of a paper "Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with GetBlunted", which will be presented at the 2021 meeting of Computability in Europe. I share primary authorship for this paper with Ryan Lorig-Roach. I am responsible for designing and partially implementing the algorithms, as well as writing the majority of the paper. Ryan Lorig-Roach also partially implemented the algorithm, and Melissa M. Meredith performed the experimental comparison to comparable algorithms.

8.2 Introduction

Genome assembly is the process of determining a sample's full genome sequence from the error-prone, fragmentary sequences produced by DNA sequencing technologies. Sequence graphs have a long history of use in this field [142, 143, 159]. In these graphs, nodes are labeled with sequences derived from sequencing data, and edges indicate overlaps between observed sequences, which may in turn indicate adjacency in the sample's genome (Fig. 8.1A). The sample genome then corresponds to some walk through graph. There are several specific sequence graph articulations in wide use, including de Bruijn graphs, overlap graphs, and string graphs. They each present computational and informational trade-offs that make them better suited to certain configurations of sequencing technologies and genome complexity.

The common topological features of genome assembly graphs are driven primarily by the repetitiveness of the underlying genomes. In many species, a large fraction of the genome consists of repeats (for instance, more than 50% of the human genome [85]). Because all copies of a repeat are highly similar to each other, the corresponding nodes in the sequence graph frequently overlap each other. In contrast, the unique regions of the genome have few erroneous overlaps. These two factors tend to create graphs that consist of long non-branching paths (corresponding to the unique regions), which meet in a densely tangled core with a complicated topology (corresponding to the repeats).

Recently, sequence graphs have also emerged into prominence in the growing field of pangenomics, which seeks to analyze the full genomes of many individuals from the same species [40]. In pangenomics, sequence graphs are used to represent genomic variation between

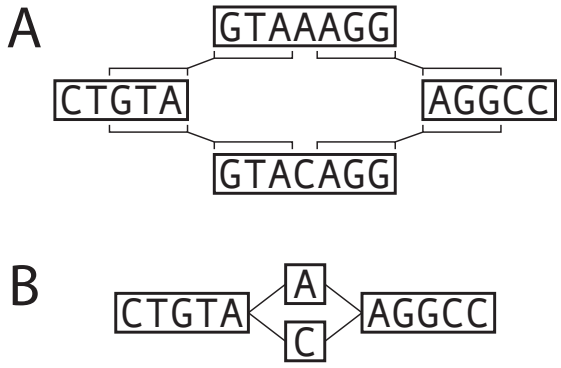


Figure 8.1: **A**: An overlapped sequence graph. **B**: A blunt sequence graph.

individual haplotypes. Sequences in the graph furcate and rejoin around sites of variation so that each individual genome corresponds to a walk through the graph (Fig. 8.1B). The growth of pangenomics has fueled major advances in both formal algorithms research [94, 172] and practical genomics tools [74, 173].

Pangenome graphs have much simpler topologies than genome assembly graphs. Having fuller knowledge of the constituent genomes makes it possible to distinguish different copies of a repeat. Thus, pangenome graphs tend to be mostly non-branching, much like the portions of assembly graphs that correspond to unique sequences in the genome. Moreover, most of the branching in pangenome graphs consists of localized bubble-like motifs. In contrast to assembly graphs, pangenome graphs have few if any cycles.

Intuitively, the shared basis in sequence graphs should permit the advances in pangenomics to spill over into genome assembly. However, such cross-pollination is stymied by a small difference in the graph formalisms. The edges in assembly graphs indicate sequence overlaps, which are necessary because of the uncertain adjacencies in the underlying genome.

In pangenome graphs, the underlying genomes are known, and the edges are *blunt* in that they indicate direct adjacency with no overlap. Blunt sequence graphs can be trivially converted into overlap graphs (with overlaps of length 0), but the reverse requires nontrivial merging operations between the overlapping sequences. As a result, methods have remained siloed within pangenomics despite potential uses in genome assembly.

In this work, we present a method to transform an overlapped sequence graph into a blunt sequence graph. We state the formal guarantees of our formulation and discuss their computational complexity. We then present an algorithm and compare its results to similar methods.

8.3 Problem statement

In transforming an overlapped sequence graph to a blunt one, we seek to provide two guarantees:

1. All walks in the overlapped graph are preserved in the blunt graph.

2. Every walk in the blunt graph corresponds to some walk in the overlapped graph.

These two properties prohibit the intuitive solution of transitively merging all overlapped sequences. Doing so can result in walks that are not present in the overlapped graph, because walks can transition between nodes that are not connected by an edge via the transitively merged sequences (Fig. 8.2). Because overlapped sequences cannot be fully merged, it is necessary to retain multiple copies of some sequences in the blunt graph. However, excessive duplication

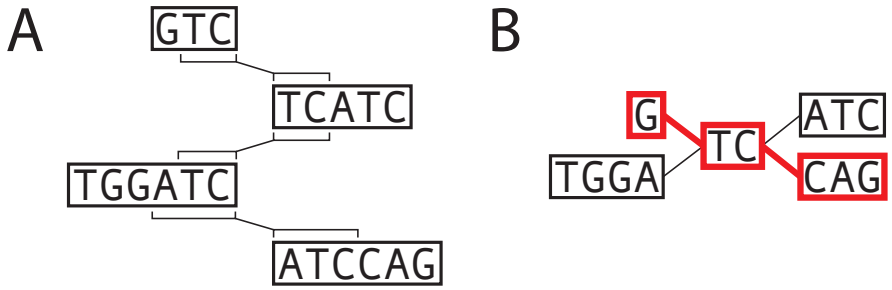


Figure 8.2: **A**: An overlapped sequence graph, and **B**: the blunt sequence graph that results from transitively merging its overlaps. The highlighted walk in the blunt graph does not correspond to any walk in the original overlapped graph.

can create problems for downstream analysis, for instance by increasing alignment uncertainty. Thus, we add one further criterion to the above formulation:

3. Minimize the amount of duplicated sequence.

8.4 Notation

An overlapped sequence graph consists of a set of sequences S and a set of overlaps $O \subset (S \times \{+,-\} \times S \times \{+,-\})$. In this notation, the symbols $+$ and $-$ indicate whether the overlap involves a prefix or suffix (collectively *affix*) of the sequence. This makes the overlapped graph a bidirected graph.

In a bidirected graph, a *walk* consists of a sequence of nodes $s_1 s_2 \ldots s_N$, $s_i \in S$ such that 1) each pair of subsequent nodes is connected by an overlap and 2) if s_{i-1} and s_i are connected by an overlap on s_i's prefix, then s_i and s_{i+1} are a connected by an overlap on s_i's suffix (or

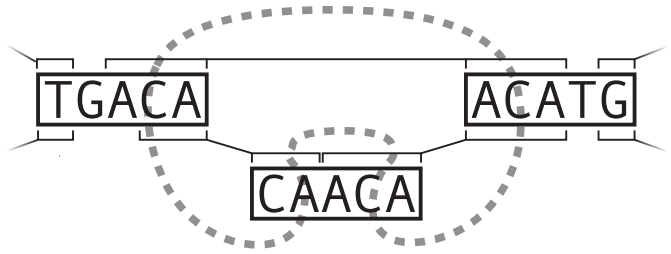


Figure 8.3: An adjacency component in a larger sequence graph. Each of the indicated affixes can reach the others by a sequence of overlaps.

vice versa). In the case that a walk traverses a node $s \in S$ from suffix to prefix, we interpret the sequence as its *reverse complement*, which is the sequence of the antiparallel strand of the DNA molecule.

Finally, an *adjacency component* is a collection of affixes (in $S \times \{+,-\}$) that can reach each other via a sequence of adjacent overlaps in O (Fig. 8.3). This sequence need not form a valid bidirected walk.

8.5 Methods

To minimize the amount of duplicated sequence, overlapped sequences must be merged. However, we have already mentioned that our criteria prohibit transitively merging all overlaps. We must then minimize the total number of groups within which overlaps are merged transitively, which coincides with the number of times the sequences need to be duplicated.

Consider a group of overlaps that contains $(s_1, s_2, +, -)$ and $(t_1, t_2, +, -)$. For merging to not introduce any walks that are not in the overlapped graph, the overlaps $(s_1, t_2, +, -)$ and

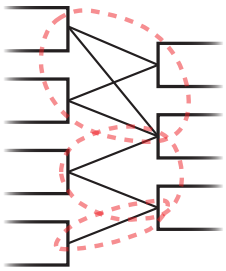


Figure 8.4: A biclique cover of an adjacency component with three bicliques.

$(t_1, s_2, +, -)$ must also be overlaps in O. Extending this logic, the entire group of overlaps must be contained within a *biclique* subgraph of the adjacency component: two sets of affixes B_1 and B_2 such that every affix in B_1 is connected to every affix in B_2 by an overlap. Thus, we can minimize the number of duplicated sequences by minimizing the number of bicliques needed to cover every overlap edge.

The problem of covering edges with the minimum number of bicliques is known as *biclique cover* (Fig. 8.4), and it is known to be NP-hard [151]. However, there are domain-specific features of overlapped sequence graphs that often make it tractable to solve large portions of the graph optimally.

First, many adjacency components are bipartite. Consider the case that an adjacency component is not bipartite, in which case there is cycle of overlaps between affixes with odd parity. Each overlap indicates high sequence similarity, so an odd cycle means that each sequence is similar to itself, reverse complemented an odd number of times. Such sequences are called DNA palindromes, and they do exist in nature. However, they comprise a small fraction of most real genomes.

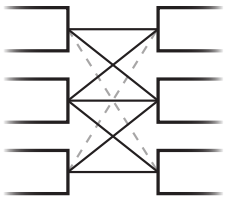


Figure 8.5: The domino graph. If either of the dotted edges are present, the induced subgraph is not a domino.

Second, most adjacency components are *domino-free*. This property refers to the absence of a particular induced subgraph, the *domino* (Fig. 8.5). A sufficient condition to prohibit dominoes is for overlapping to be a transitive property. That is, whenever sequence s_1 overlaps sequences t_1 and t_2, and sequence s_2 overlaps t_1, then s_2 also overlaps t_2. In reality, this is not always the case. However, it is very often the case, since overlaps indicate sequence similarity, and similarity is approximately transitive.

These features guided the design of the following algorithm. If an adjacency component is bipartite and domino-free, we compute the biclique cover in polynomial time with the algorithm of Amilhastre, Vilaren, and Janssen [7]. When an adjacency component is bipartite but not domino-free, we instead use the dual graph reduction algorithm of Ene, et al. [64], followed by their lattice-based post-processing if the algorithm does not identify the optimal solution. Finally, if an adjacency component is not bipartite, we first reduce it to the bipartite case by computing an approximate solution to the maximum bipartite subgraph problem using the algorithm of Bylka, Idzik, and Tuza [26]. The maximum bipartite subgraph problem is equivalent to max cut, which is also NP-hard [99]. This process is repeated recursively on the

edges that are not included in the bipartite subgraph.

The amount of duplicated sequence is also affected by the manner in which sequences are merged among the overlaps of a biclique. To minimize duplicated sequence, we must maximize matches in the alignment between the overlapped sequences. This is the multiple sequence alignment problem, which is NP-hard. We use the partial order alignment algorithm to approximate the optimal multiple sequence alignment [111]. Partial order alignment also has the advantage that the alignment is expressed as a blunt sequence graph, which can be directly incorporated in the full blunt graph.

8.6 Implementation

We have implemented the algorithm described here as a genomics tool called GetBlunted. GetBlunted takes as input a GFA file (a common interchange format for sequence graphs [122]) and outputs a GFA containing a blunt graph. In addition, it provides a translation table from sequences in the output to sequences in the input, which can be used to translate analyses performed on the blunt graph into analyses on the overlapped graph. The implementation is written entirely in C++, and it use several auxiliary libraries: GFAKludge is used for manipulating GFA files [50], libbdsg is used to represent sequence graphs [62], and SPOA is used for partial order alignment [210].

8.7 Results

We compared the performance of GetBlunted to two other tools that transform overlapped sequence graphs into blunt graphs: the gimbricate/seqwish [71, 72] pipeline and Stark [148]. These are, to our knowledge, the only other such tools besides GetBlunted. However, they are not completely comparable. Neither tool provides the guarantees that GetBlunted does for preserving the walk space of the graph. In addition, Stark only works with de Bruijn graphs, a restricted subset of overlap graphs in which all overlaps are exact matches of a uniform length.

We profiled speed and memory usage on three assembly graphs. The first two are assembly graphs produced by the Shasta assembler [185] for the haploid human cell line CHM13 and for human sample HG002. Both of these were built using Oxford Nanopore reads[1]. The last graph is a de Bruijn graph of Pacific Biosciences HiFi reads of an *Escherichia coli* strain (SRR10382245), which was constructed using jumboDB [15].

All of the bluntifying tools were run on a single core of a c5.9xlarge AWS instance with an Intel Xeon Scalable Processor. Memory usage and compute time were measured with the Unix time tool. The results of the profiling are presented in Table 8.1. GetBlunted is over 1000 times faster than and comparably memory-intensive to the gimbricate/seqwish pipeline. For de Bruijn graphs, Stark is faster than either tool, although this performance comes at the cost of limited generality.

[1] Publicly available at `https://s3-us-west-2.amazonaws.com/miten-hg002/index.html?prefix=guppy\_3.6.0/`

Assembly	Bluntification Tool	Run Time (min)	RAM (GB)
HG002 Shasta	GetBlunted	0.35	9
	gimbricate/seqwish	917.5	6
CHM13 Shasta	GetBlunted	0.38	4
	gimbricate/seqwish	314.6	6
E. coli de Bruijn	GetBlunted	8.36	26
	gimbricate/seqwish	10.74	4
	Stark	0.65	3

Table 8.1: Table of speed and memory usage of bluntifing tools run on a single core of an AWS server.

8.8 Discussion

In this work, we described an algorithm and software tool, GetBlunted, which transforms overlapped sequence graphs into blunt sequence graphs. This provides a route for sequence graph methods developed for pangenomics to be applied to sequence graphs in genome assembly. In both fields, walks through the sequence graph are of primary importance. In genome assembly, some walk through the graph corresponds to the sample genome. In pangenomics, the genomes used to construct the pangenome each correspond to a walk through the graph. GetBlunted provides attractive guarantees that it faithfully preserves the walk space of the input while also producing parsimonious output. Other comparable methods either do not provide these guarantees or only provide them in limited cases. In addition, GetBlunted is (except in the case of de Bruijn graphs) faster than alternatives that do not provide these guarantees, and it has resource requirements that are easily achievable in any computational environment that is used for genome assembly. In the future, GetBlunted could serve as an step in genome assembly pipelines to improve the quality of their overlap graphs. It could also facilitate direct

analyses of assembly graphs in metagenomics applications.

Part V

Discussion

Chapter 9

Discussion

For many years, progress in pangenomics was hampered by ascertainment bias in genomic variation assays. In order to comprehensively identify minor alleles, sequencing projects needed to be carried out at population scale. The only genome inference technologies that were economical at this scale were genotype arrays and NGS variant calling. Accordingly, scientific knowledge of variation shared these assays' biases. The ascertained variants were disproportionately small and disproportionately located in high-entropy, unique regions of the genome. Almost by definition, these are the variants that were least likely to have been substantially impacted by reference bias to begin with.

This situation is now beginning to change. Advancements in long read assembly methods are shining a light on previously hidden regions of the genome. This can be seen in the first complete human genome assembly by the T2T Consortium [149] or in the intensive characterization of structural variation by the Human Genome Structural Variation Consortium (HGSVC) [31, 57]. The Human Pangenome Reference Consortium (HPRC) is currently en-

gaged in a similar assembly effort [83]. These data resources are already being used in pangenomics applications to genotype structural variation with NGS data [58, 191], a poster child for genomics tasks that were previously frustrated by severe reference bias.

The research presented in this dissertation sits somewhere on the threshold between these two periods of pangenomics. In chapter 7, I helped establish foundational theory for pangenome graphs. Chapters 5 and 6 detail engineering efforts that underlie the recent emergence of practical and usable pangenomics. In chapter 4, I developed a backward linkage from graph-based pangenomics to conventional reference analyses. Finally, in chapters 3 and 8, I contributed toward extending the concepts and methods developed in pangenomics to other areas of genomics: transcriptomics and genome assembly respectively.

The new genome assembly data resources from the HGSVC and HPRC are likely to open new challenges for pangenomics method developers. Up until now, most pangenome graphs have been constructed by augmenting a reference genome with variant alleles [74, 103]. As the field pivots toward genome assemblies, it will increasingly be necessary to construct graphs using whole genome multiple sequence alignments [11, 120, 139]. In fact, the HPRC is also working to produce graphs with this methodology. Such graphs have comparatively complicated structures, with cyclic motifs and regions of high structural complexity. The assumption of structural simplicity is subtly baked into the design of many existing pangenomics applications, and it is likely that many tools' performance will decline on these new graphs—if they work at all. However, these challenges also represent opportunities. Much of the complexity in assembly-based graphs is due to true complexity in the organization of genomic variation, and the formalism must be rich enough to express this variation in order to analyze it.

The linkages between pangenomics and population genomics remain tight. This is evident in the discussion of pangenomics data resources above, which could just as easily have been a discussion of population genomics data resources. The challenges and opportunities are similarly mirrored. Population genomics remains, on the whole, invested in a variant-centered analysis framework, but the concepts and tools being developed in pangenomics offer a tantalizing suggestion of a more general approach. A pangenome graph can be thought of as a map of homologies between a collection of haplotypes. In this sense, it implicitly defines an approximation to the ancestral recombination graph (ARG) with a similar flavor to the Li and Stevens model [125]: the relationships between haplotypes are explicit but without any overlaid ancestry-based tree structure. However, in another sense the pangenome graph represents a *more* exact formalism to express the ARG than the sequence-of-trees that is commonly used for variant data [101, 168]. This is because the sequence-of-trees inherits all of the simplifications and biases of the reference-based variant calling methodology. The richer structure of homologies in a pangenome graph hints at the possibility of advancing closer to the theoretical ARG holy grail: an ancestral history of every base in a recombining, (structurally) mutating genome. Doing so will probably require introducing ancestry-based tree structures to the graph's haplotypes, as has been done in theoretical studies of genome evolution [156].

I will suggest that the most productive route forward in pursuing the synergies between population genomics and pangenomics is to be guided by applications. Two recently published pangenomics tools strike me as exemplifying this approach. The danbing-tk tool used lossy, compressive pangenome graph structures to characterize variable nucleotide tandem repeats (VNTRs) [128]. These loci have high enough polymorphism that more exact graphs

would be difficult to build in practice, and the lossy graphs could still support dosage models of the genotypes that exhibited population structure and phenotypic associations. Another tool, PanGenie, boosts its structural variant calling accuracy by imputing genotypes with a model that essentially makes explicit the pangenome graph's implicit Li and Stevens model mentioned above [58]. Both of these studies display a pragmatic balance between computational difficulty, complexity of the pangenome formalism, and the population genomics analyses they want to support. However, I hope that pragmatism will not forestall attempts to analyze features of the ARG that necessitate the richer conceptual framework offered by pangenomics.

Moving forward, I predict that there will be a trend of applying pangenomics methodologies to increasingly many areas in functional genomics as well. My work on haplotype-resolved transcriptomics in Chapter 3 is a step forward in this direction. It offers an example of how both the technical formalisms of pangenomics and the mitigation of reference bias can be applied to a functional assay. Work has also been done on histone modifications [79] and transcription factor binding [80] by other groups. Plenty of areas remain: 3D genome organization, DNA accessibility, perturbation screens, DNA and RNA modifications, somatic mutations, etc. It remains to be seen which of these areas will stand to benefit the most from pangenomics approaches. In exploring applications in these domains, I believe there is a danger of becoming overly focused on sequence graph methods as such. For instance, it is not at all clear to me that sequence graphs are a useful formalism for a 3D organization pangenome. As methods developers we should be guided by the problem rather than the familiarity of current approaches. Ultimately, broad adoption in a field will depend both on how troublesome reference bias is and on the usefulness of novel conceptualizations of population genomics.

Another area of pangenomics that I expect to be very active in the future is annotation. Genome annotation is a vital function for conventional reference genomes, but relatively little work has been done on pangenome annotation. I suggest that it is useful to differentiate between two classes annotations for pangenomes, which I refer to as *proper pangenome annotations* and *conditional genome annotations*. By proper pangenome annotations, I mean annotation with data that cannot be simply or unambiguously expressed relative to any particular haplotype. Examples include intraspecific sequence conservation or variant effects, both of which are implicitly annotations of multiple haplotypes. By conditional genome annotations, I mean annotation with data that is most meaningful when expressed relative to one haplotype but nevertheless shows population variation. Because of this variation, conditional annotations may benefit from using pangenomics as an organizing structure. Examples could include gene transcripts and regulatory elements (for which a similar concept has already been applied [205]). I expect that the coexistence of these classes of annotation will necessitate informatics and visualization tools that can both express annotations of entire pangenome graphs and perform on-demand queries for annotations conditioned on a specific haplotype. In the latter case, I further expect that the tools of comparative annotation will provide a productive route forward [10].

Pangenomics approaches are already being applied outside of human genomics. In fact, they were applied much earlier in microbes [134], although not with nucleotide resolution until recently [36]. Recent studies have also investigated other multicellular eukaryotes with a clear emphasis on agricultural plants: *A. thaliana* [5], cabbage [75], rice [163, 200, 219], cow [46–48], tomato [68], soybean [126], rapeseed [195], wheat [213], cotton [124], and eggplant [17]. As this march continues, I believe it is a good time to begin thinking about what the

points of contact will be between pangenomics and comparative genomics. Indeed, many of the technical tools of comparative genomics (notably whole genome alignment) are already being repurposed for pangenomics [11]. I expect that the greatest gains are likely to be found in areas where there is already fruitful cross-pollination between comparative genomics and population genomics, such as hybridization and incomplete lineage sorting. In any case, the barriers to such analysis seem greater than for intraspecific analysis, which I expect to be the mainstay of pangenomics for the near future.

I do not share the optimism of some early adopters that pangenome references will supplant conventional reference genomes, which maintain a substantial advantage in efficiency and conceptual simplicity. For many problems in genomics, conventional references, for all their simplifications and inaccuracies, are perhaps not great but at least good enough. However, I do expect reference pangenomes to become an important public data resource. The task of constructing a quality pangenome is challenging, and the genomics community stands to benefit greatly by not having to repeat it.

It also seems likely that a reference pangenome may never have the air of finality that the genomics community has invested reference genomes with. Any map of homologies is bound to be incomplete and erroneous, which means there will always be uncertainty (and probably contest) over the correct structure for the pangenome. However, it is worth noting that assembling a large eukaryotic genome once seemed similarly insurmountable, and yet the reference genome still attained the venerated role that it now occupies. The elusiveness of that sense of finality for a reference pangenome may be a simple matter of time. Alternatively, it may stem from the fact that the project of pangenomics began by centering the problem of

reference bias and thereby, in a sense, admitting its own weaknesses.

In any case, this is an exciting time for pangenomics. In the last few years, pangenomics methods have become increasingly practical and mainstream. Moreover, I believe strongly that many areas of genomics are ripe for pangenomic analysis, as I hope this discussion has communicated. I am confident that the pangenomics community is capable of moving this program of research forward, or maybe reverse complement.

Part VI

Appendices

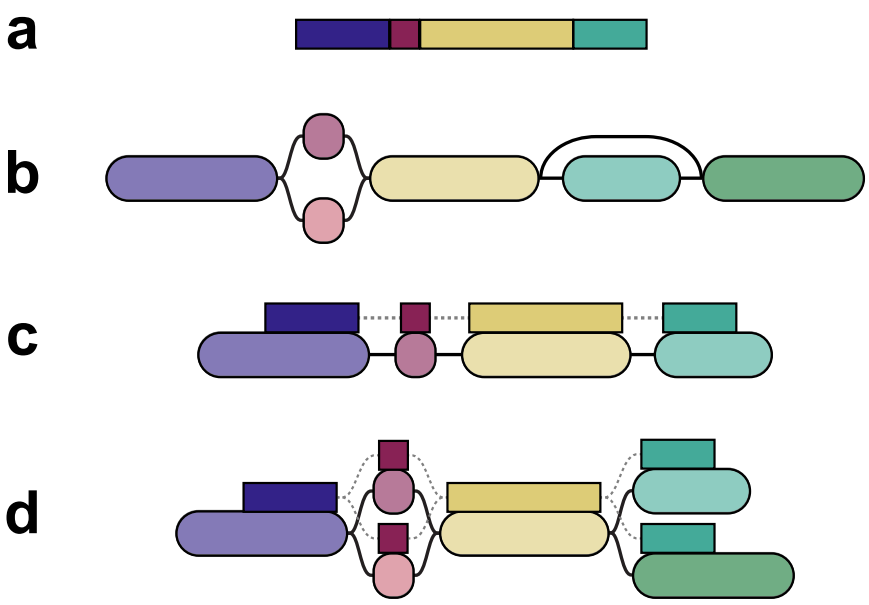
Appendix A

Appendix A: Supplementary information for pantranscriptome paper

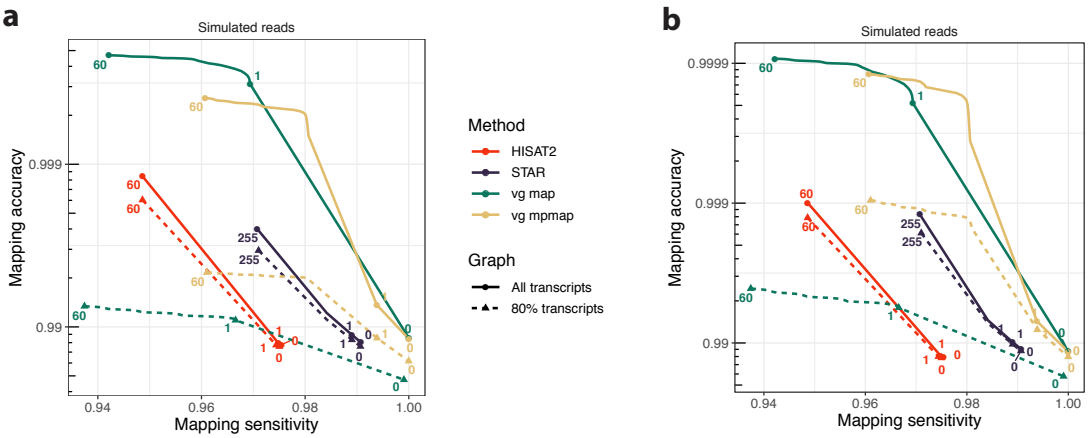
A.1 Preamble

This Appendix contains the supplementary information included with the preprint for “Haplotype-aware pantranscriptome analyses using spliced pangenome graphs”, which is included in this dissertation as Chapter 3.

A.2 Supplementary Figures

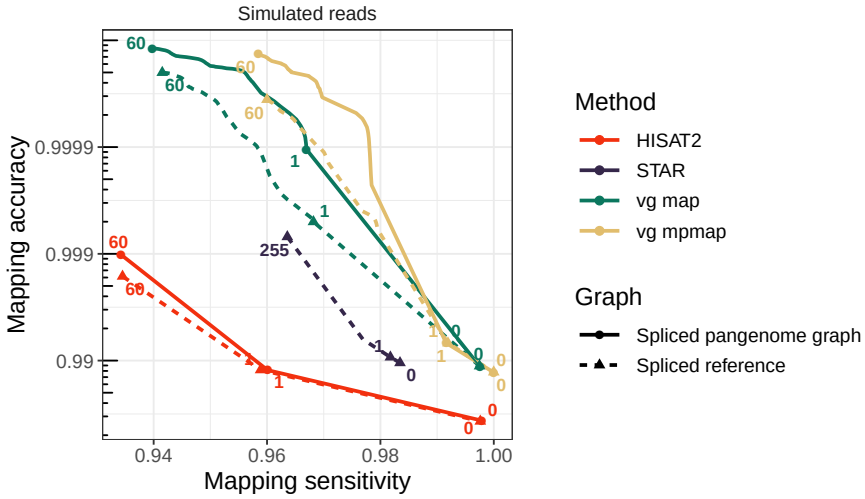


Supplementary Figure A.1: Diagram of a multipath alignment
A diagrammatic comparison between the multipath alignment output of vg mpmap and the single-path alignment output of other graph aligners (such as vg map). **a** A read and **b** a sequence graph, which have been colored to indicate which parts of the read could plausibly align to which parts of the graph. **c** A single-path alignment. The read sequence is aligned to one path from the graph. **d** A multipath alignment. The alignment can split and rejoin to express the alignment uncertainty to different paths in the graph.



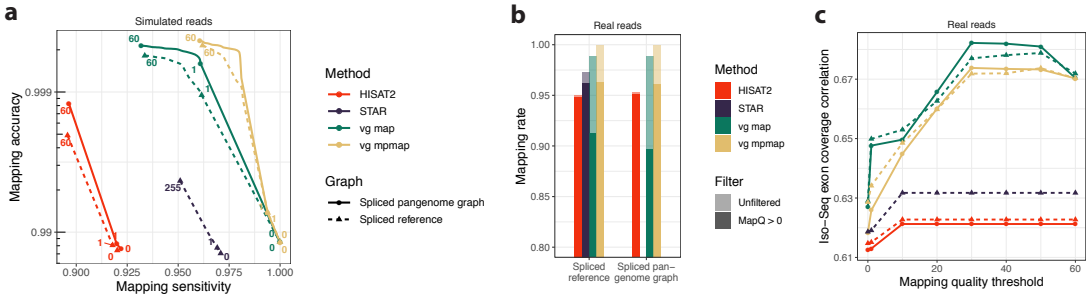
Supplementary Figure A.2: Mapping benchmark to novel splice-junctions using RNA-seq data from NA12878

RNA-seq mapping results comparing vg mpmap against three other methods using simulated Illumina data ("vg sim (ENC, uniform)" in Supplementary Table A.4). Shows mapping accuracy and sensitivity for different mapping quality thresholds (colored numbers). An alignment is considered correct if it covers 90% (**a**) or 70% (**b**) of the true reference sequence alignment. Solid lines show the results using a spliced pangenome graph (spliced reference for STAR) generated using the complete transcript annotation. Dashed lines show the results using a reference generated with a random subset of 80% of the transcripts in the annotation.



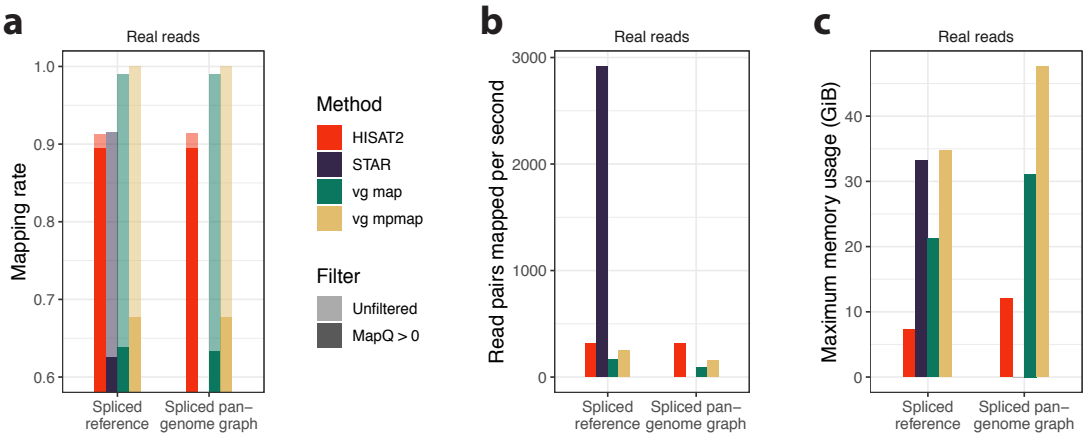
Supplementary Figure A.3: Graph-based mapping benchmark using RNA-seq data from NA12878

Mapping accuracy and sensitivity for vg mpmap and three other methods using simulated Illumina data ("vg sim (ENC, uniform)" in Supplementary Table A.4). Colored numbers indicate different mapping quality thresholds. An alignment is considered correct if its start position is within 100 bases from the start position of the true alignment measured using any labeled transcript path in the graph or the linear reference sequence. Solid and dashed lines show the results using a spliced pangenome graph and spliced reference, respectively.



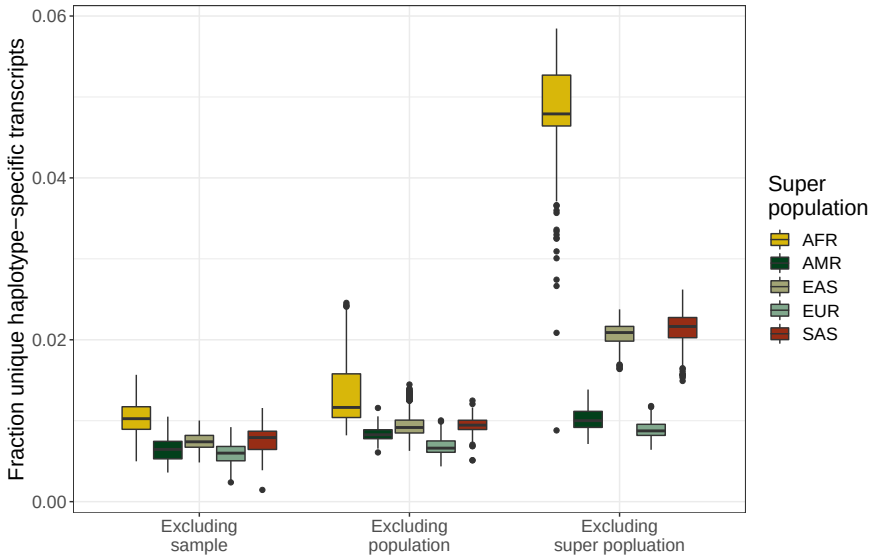
Supplementary Figure A.4: Mapping benchmark using RNA-seq training data from NA12878

RNA-seq mapping results comparing vg mpmap and three other methods using the simulated and real Illumina data that was used in the optimization of vg map and vg mpmap ("vg sim (SRR, uniform)" and "SRR1153470" in Supplementary Table A.4 and A.3, respectively). Solid and dashed lines show the results using a spliced pangenome graph and spliced reference, respectively. **a** Mapping accuracy and sensitivity for different mapping quality thresholds (colored numbers) using simulated data. An alignment is considered correct if it covers 90% of the true reference sequence alignment. **b** Mapping rate using real data. The shaded bars show the mapping rate for all alignments and the solid bars for only alignments with a mapping quality above 0. **c** Pearson correlation between Illumina and Iso-Seq exon coverage using real data as a function of mapping quality threshold. Exons are defined by the Iso-Seq alignments.



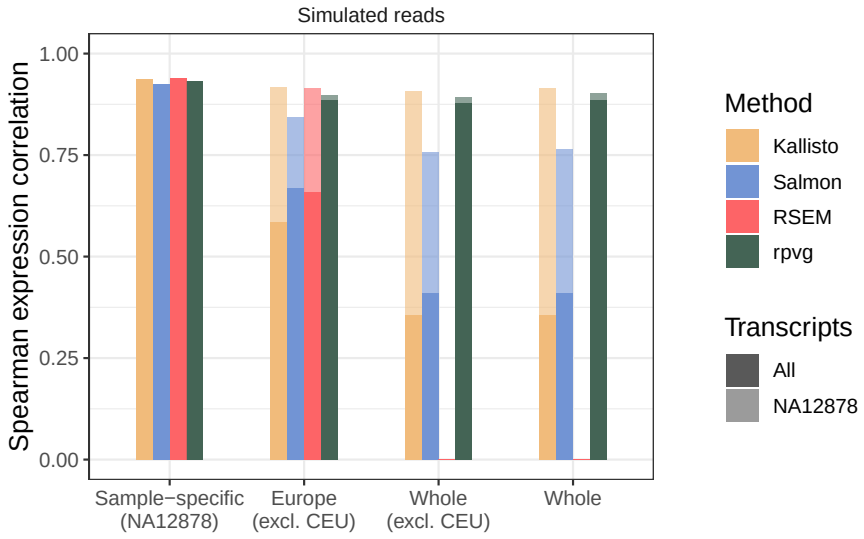
Supplementary Figure A.5: Mapping benchmark using RNA-seq training data from CHM13

RNA-seq mapping results comparing vg mpmap against three other methods using real Illumina data that was used in the optimization of vg mpmap ("CHM13" in Supplementary Table A.3). **a** Mapping rate. The shaded bars show the mapping rate for all alignments and the solid bars for only alignments with a mapping quality above 0. **b** Number of read pairs mapped per second per thread. The mapping times were measured using 16 threads on a AWS m5.4xlarge instance. **c** Maximum memory usage for mapping in gigabytes.



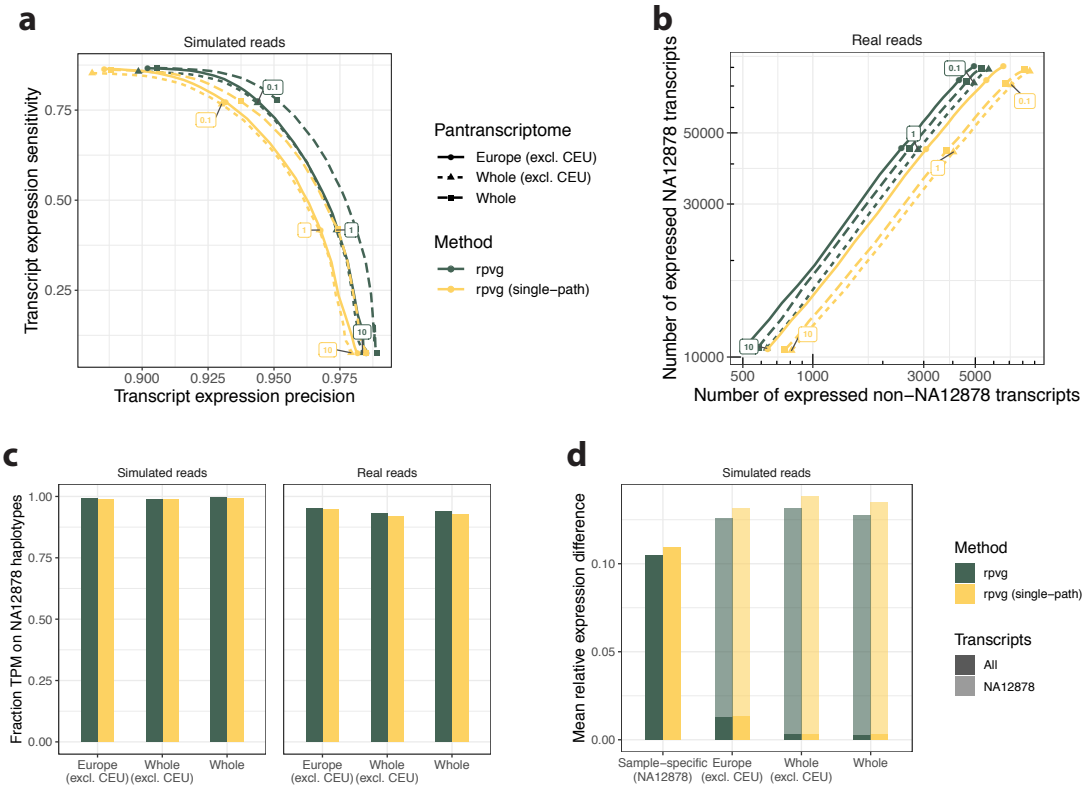
Supplementary Figure A.6: Haplotype-specific transcript uniqueness in a 1000 Genomes Project pantranscriptome

The fraction of HSTs that are unique to each sample in the 1000 Genomes Project (1000GP) when compared to different subsets of samples in the 1000GP. Left box plots show the fraction unique when comparing to all other samples, middle box plots show the fraction unique when comparing to all other samples excluding the samples' population, and right box plots show the fraction unique when comparing to all other samples excluding the samples' super population. AFR: African, AMR: Admixed American, EAS: East Asian, EUR: European, SAS: South Asian.



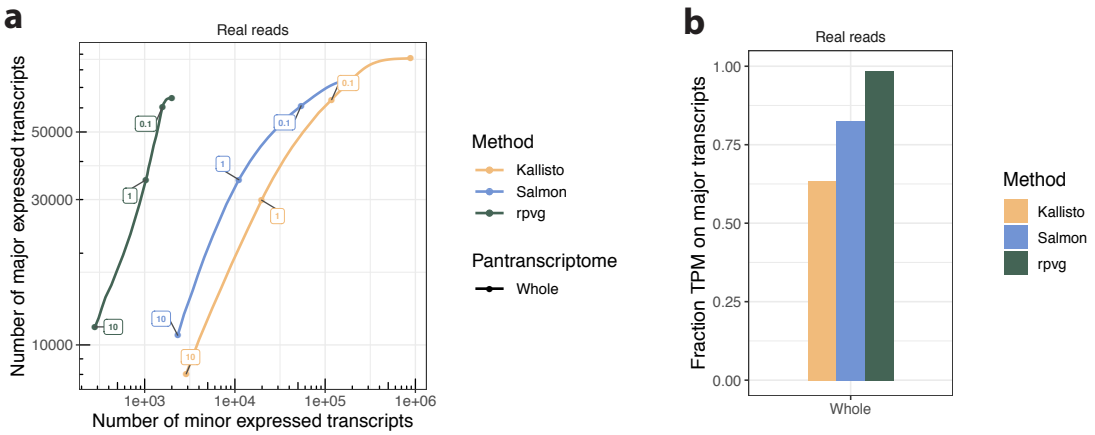
Supplementary Figure A.7: Haplotype-specific transcript expression correlation benchmark using RNA-seq data from NA12878

Haplotype-specific transcript (HST) quantification results comparing rpvg and three other methods using simulated Illumina data ("vg sim (ENC, RSEM)" in Supplementary Table A.4). Shows Spearman correlation between simulated and estimated expression (in transcripts per million (TPM)) for different pantranscriptomes. Correlation was calculated using either all HSTs in the pantranscriptome (solid bars) or using only the NA12878 HSTs (shaded bars). "Sample-specific (NA12878)" is a personal transcriptome generated from 1000 Genomes Project (1000GP) NA12878 haplotypes. "Europe (excl. CEU)" is a pantranscriptome generated from European 1000GP haplotypes excluding the CEU population. "Whole (excl. CEU)" and "Whole" are pantranscriptomes generated from all 1000GP haplotypes without and with the CEU population, respectively.



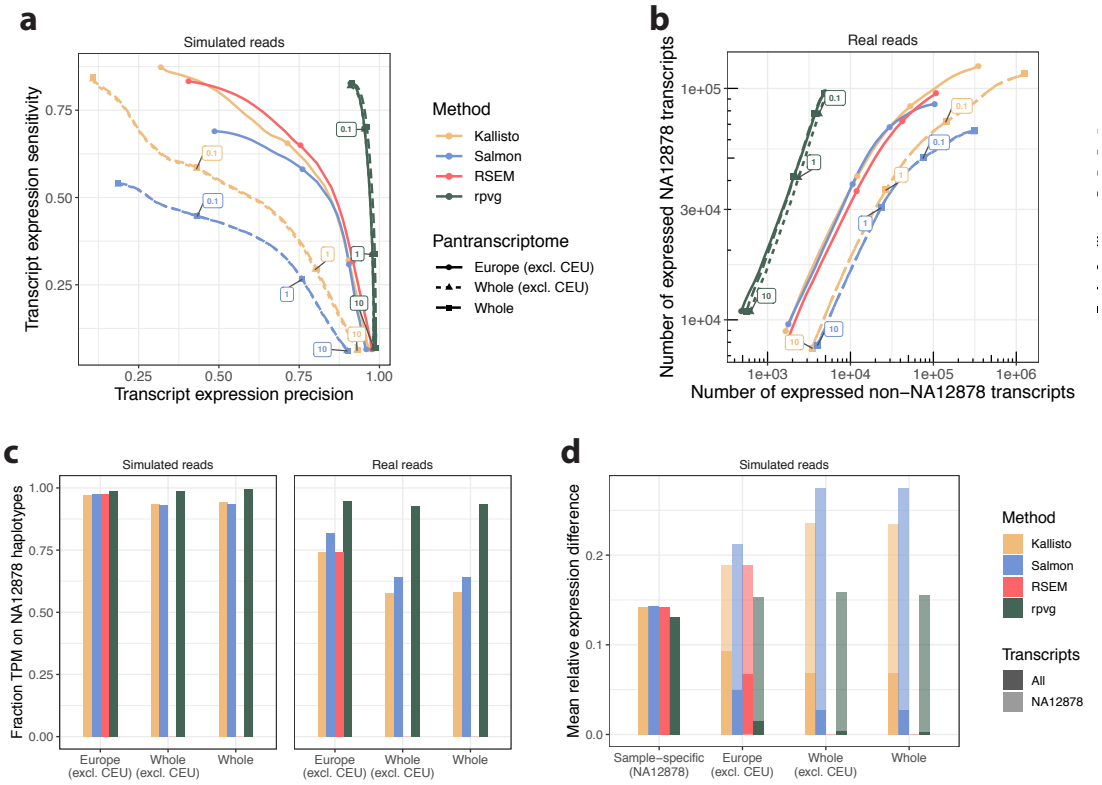
Supplementary Figure A.8: Multipath alignment benchmark using RNA-seq data from NA12878

Haplotype-specific transcript (HST) quantification results comparing rpvg with single-path and multipath alignments using simulated and real Illumina data ("vg sim (ENC, RSEM)" and "ENCSR000AED" in Supplementary Table A.4 and A.3, respectively). Solid lines with circles are results using a pantranscriptome generated from 1000 Genomes Project (1000GP) European haplotypes excluding the CEU population. Dashed lines with triangles and squares are results using a pantranscriptome generated from all 1000GP haplotypes without and with the CEU population, respectively. The single-path alignments were created by finding the best scoring path in each multipath alignment. **a** Sensitivity and precision of whether a transcript is correctly assigned nonzero expression for different expression value thresholds (colored numbers for "Whole (excl. CEU)" pantranscriptome) using simulated data. Expression is measured in transcripts per million (TPM). **b** Number of expressed transcripts from NA12878 haplotypes shown against the number from non-NA12878 haplotypes for different expression value thresholds (colored numbers) using real data. **c** Fraction of transcript expression (in TPM) assigned to NA12878 haplotypes for different pantranscriptomes using simulated (left) and real (right) data. **d** Mean absolute relative difference (MARD) between simulated and estimated expression (in TPM) for different pantranscriptomes using simulated data. MARD was calculated using either all HSTs in the pantranscriptome (solid bars) or using only the NA12878 HSTs (shaded bars). "Sample-specific (NA12878)" is a personal transcriptome generated from 1000GP NA12878 haplotypes.



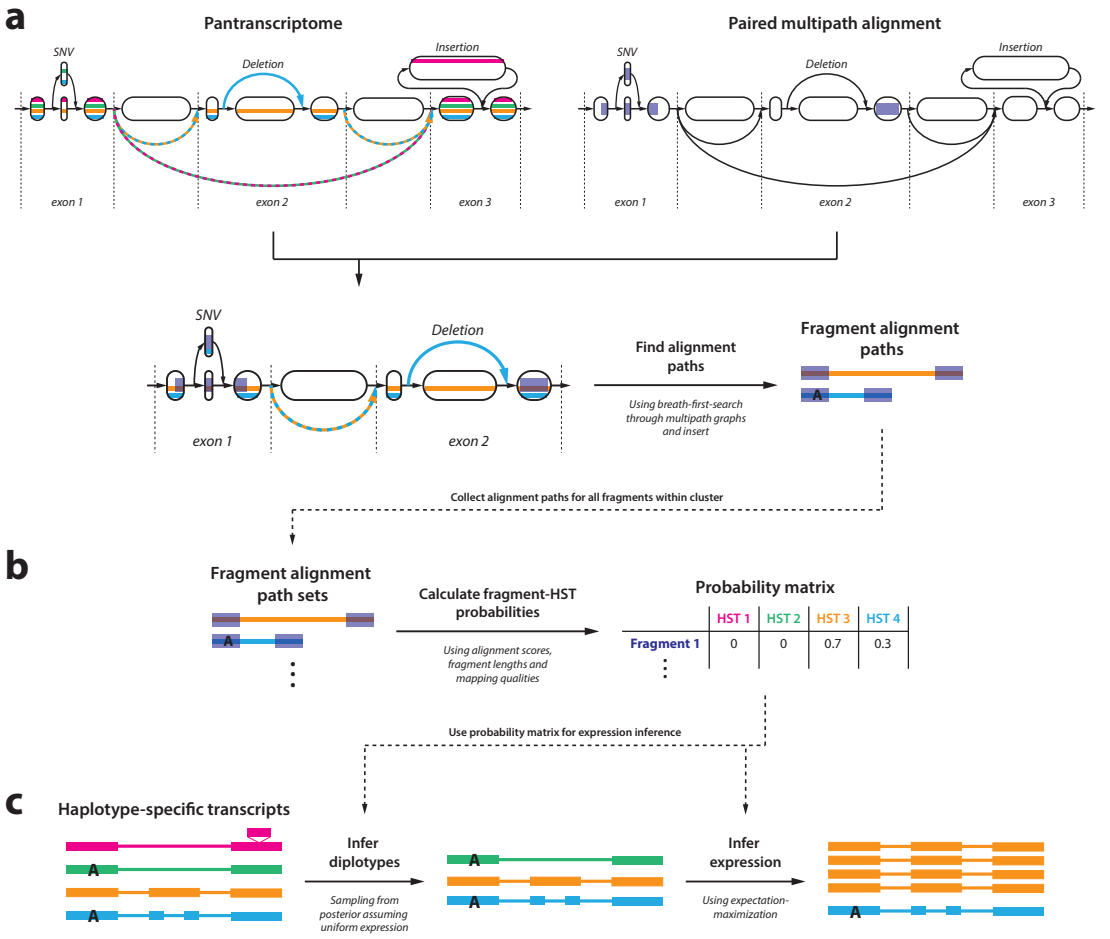
Supplementary Figure A.9: Haplotype-specific transcript quantification benchmark using RNA-seq training data from CHM13

Haplotype-specific transcript (HST) quantification results comparing rpvg against two other methods using real Illumina data that was used in the optimization of rpvg ("CHM13" in Supplementary Table A.3). All experiments used a pantranscriptome generated from all 1000 Genomes Project (1000GP) haplotypes. Each HST is either classified as major or minor. Major HSTs are defined as the highest expressed haplotype for each transcript; the rest are defined as minor. As CHM13 is effectively haploid, the fraction of expression from minor HSTs is a lower bound on the fraction of incorrectly inferred transcript expression. **a** Number of major expressed transcripts against the number of minor expressed for different expression value thresholds (colored numbers). Expression is measured in transcripts per million (TPM). **b** Fraction of transcript expression (in TPM) assigned to major transcripts for different methods.



Supplementary Figure A.10: Haplotype-specific transcript quantification benchmark using RNA-seq training data from NA12878

Haplotype-specific transcript (HST) quantification results comparing rpvg against three other methods using simulated and real Illumina data that was used in the optimization of rpvg ("vg sim (SRR, RSEM)" and "SRR1153470" in Supplementary Table A.4 and A.3, respectively). Solid lines with circles are results using a pantranscriptome generated from 1000 Genomes Project (1000GP) European haplotypes excluding the CEU population. Dashed lines with triangles and squares are results using a pantranscriptome generated from all 1000GP haplotypes without and with the CEU population, respectively. **a** Sensitivity and precision of whether a transcript is correctly assigned nonzero expression for different expression value thresholds (colored numbers for "Whole (excl. CEU)" pantranscriptome) using simulated data. Expression is measured in transcripts per million (TPM). **b** Number of expressed transcripts from NA12878 haplotypes shown against the number from non-NA12878 haplotypes for different expression value thresholds (colored numbers) using real data. **c** Fraction of transcript expression (in TPM) assigned to NA12878 haplotypes for different pantranscriptomes using simulated (left) and real (right) data. **d** Mean absolute relative difference (MARD) between simulated and estimated expression (in TPM) for different pantranscriptomes using simulated data. MARD was calculated using either all HSTs in the pantranscriptome (solid bars) or using only the NA12878 HSTs (shaded bars). "Sample-specific (NA12878)" is a personal transcriptome generated from 1000GP NA12878 haplotypes.



Supplementary Figure A.11: Diagram of haplotype-specific transcript quantification in rpvg

Diagram showing an overview of how rpvg infers expression of haplotype-specific transcripts (HSTs) in a pantranscriptome from a set of paired-end multipath alignments (see Supplementary Figure 10). The colored thin lines correspond to HST paths, and the blue transparent regions correspond to aligned read sequences. **a** For each fragment, all paths through the multipath alignment graphs are identified using a depth-first-search (DFS). For paired-end reads, the DFS also traverses the fragment insert creating alignment paths of the whole fragment. Only alignment paths that follow an HST path in the pantranscriptome are kept. **b** The probabilities that each fragment originated from each of the HSTs in a cluster are calculated using the score and length of the fragment alignment paths, and the mapping quality. **c** The fragment-HST probability matrix is used to infer the expression of the HSTs using a nested inference scheme. First, a distribution over diplotypes (if sample is diploid) is inferred. A haplotype combination is then sampled from this distribution and expression is inferred conditioned on the sampled haplotypes using expectation-maximization. This procedure is repeated a 1,000 times to account for the uncertainty in the haplotype estimates.

A.3 Supplementary Tables

Name	Description	Non-exonic variant filter	Number of samples	Number of exonic variants	Number of total variants
1000GP (NA12878)	1000GP variants observed in NA12878	None	1	152,968	4,266,678
1000GP (EUR, excl. CEU)	All european (EUR) 1000GP variants excluding variants unique to the CEU population	AF $\geq$ 0.002	404	986,167	15,041,731
1000GP (all, excl. CEU)	All 1000GP variants excluding variants unique to the CEU population	AF $\geq$ 0.001	2,405	3,808,242	31,731,676
1000GP (all)	All 1000GP variants	AF $\geq$ 0.001	2,504	3,873,100	29,970,512

Supplementary Table A.1: Genomic variant (haplotype) sets
1000GP: 1000 Genomes Project

Name	Transcript annotation	Number of transcripts	Haplotype set†	Number of haplotypes	Number of haplotype-specific transcripts
Sample-specific (NA12878)	GENCODE v29 (full-length)	172,449	1000GP (NA12878)	2	235,400
Europe (excl. CEU)	GENCODE v29 (full-length)	172,449	1000GP (EUR, excl. CEU)	808	2,515,408
Whole (excl. CEU)	GENCODE v29 (full-length)	172,449	1000GP (all, excl. CEU)	4,810	11,626,948
Whole	GENCODE v29 (full-length)	172,449	1000GP (all)	5,008	11,835,580

Supplementary Table A.2: Pantranscriptomes
†See Supplementary Table A.1 for more details on the haplotype sets

Name / Accession (source)	Reference	Replicate	Cell-line	Sequencing type	Number of read-pairs or alignments
SRR1153470 (SRA)	[204]	NA	NA12878	Paired-end Illumina HiSeq 2000	115,359,773 read-pairs
ENCSR000AED (ENCODE)	[44, 49]	1	NA12878	Paired-end Illumina HiSeq 2000	97,548,052 read-pairs
ENCSR706ANY (ENCODE)	[44, 49]	1-4	NA12878	PacBio Iso-Seq	2,687,717 alignments
CHM13 (T2T)	†	1	CHM13	Paired-end Illumina NovaSeq	90,930,105 read-pairs
ERR188217 & ERR204848 (Geuvadis)	[108]	1	NA11832	Paired-end Illumina HiSeq 2000	2,354,918 + 863,919 read-pairs
ERR188235 & ERR204834 (Geuvadis)	[108]	1	NA11930	Paired-end Illumina HiSeq 2000	1,305,660 + 1,220,864 read-pairs
ERR188354 & ERR204977 (Geuvadis)	[108]	1	NA12775	Paired-end Illumina HiSeq 2000	3,088,867 + 806,975 read-pairs
ERR188429 & ERR204822 (Geuvadis)	[108]	1	NA12889	Paired-end Illumina HiSeq 2000	2,286,090 + 778,671 read-pairs

Supplementary Table A.3: Read sets and alignments
†Downloaded from the T2T consortium data repository:
`https://github.com/nanopore-wgs-consortium/CHM13`

Name	Training reads†	Haplotype-specific transcripts‡	Expression values	vg sim parameters	Number of read-pairs
vg sim (SRR, uniform)	SRR1153470	Sample-specific (NA12878)	Uniform	Indel error: 0.001, mean: 277, sd: 43	50,000,000
vg sim (ENC, uniform)	ENCSR000AED	Sample-specific (NA12878)	Uniform	Indel error: 0.001, mean: 216, sd: 24	50,000,000
vg sim (SRR, RSEM)	SRR1153470	Sample-specific (NA12878)	RSEM estimates	Indel error: 0.001, mean: 277, sd: 43	50,000,000
vg sim (ENC, RSEM)	ENCSR000AED	Sample-specific (NA12878)	RSEM estimates	Indel error: 0.001, mean: 216, sd: 24	50,000,000

Supplementary Table A.4: Simulated read sets

†See Supplementary Table A.3 for more details on the training read sets

‡See Supplementary Table A.2 for more details on the haplotype-specific transcript sets

Software tool(s) or library	Used for step involved in	Version or GitHub commit(s)†
rpvg	Inferring expression.	a7a79697
vg construct, vg convert, vg rna, vg ids, vg index, vg stats & vg gbwt	Constructing graphs and pantranscriptomes.	v1.23.0, c861e23e & 8ff022c3
vg gbwt	Constructing GBWT r-index.	883f0f87
vg view & vg sim	Simulating reads.	515a4595
vg snarls, vg prune & vg index	Constructing distance and GCSA graph index.	8ff022c3 & c4bbd63b
vg map & vg mpmap	Mapping reads.	c4bbd63b
vg inject, vg paths & vg surject	Converting alignments between graph (GAM) and reference (BAM).	c4bbd63b
vg index, vg paths & vg surject	Creating reference transcript alignments for mapping benchmark.	c861e23e
vg stats, vg view & vg gampcompare	Comparing graph alignments for mapping benchmark.	c4bbd63b
vgrna-project-scripts	Converting expression profiles for simulations. Inferring coverage, overlap and bias statistics for mapping benchmark. Comparing haplotype-specific transcript sequences for expression benchmark. Plotting the results.	71442ea4 & afc94cf8

Supplementary Table A.5: Versions of internal software used
†Different subcommands in the vg toolkit and parts of the pipeline stabilized at different times during our development process, hence the variety of commits used.

Software tool(s) or library	Used for step involved in	Version or GitHub commit(s)†
bcftools	Filtering, subsetting and normalizing variants.	v1.9
samtools	Filtering, sorting, converting and indexing alignments.	v1.9
bedtools	Converting alignments to regions and calculating coverage.	v2.29.1
seqtk	Subsampling reads.	v1.3
HISAT2	Indexing graphs and mapping reads.	v2.2.0 & v2.2.1
STAR	Indexing references and mapping reads.	v2.7.3a
Bowtie2	Indexing pantranscriptomes and mapping reads.	v2.3.5.1
RSEM	Indexing pantranscriptomes and inferring expression.	v1.3.1
Kallisto	Indexing pantranscriptomes and inferring expression.	v0.46.1
Salmon	Indexing pantranscriptomes and inferring expression.	v1.2.1
SeqLib	Parsing alignments and calculating overlaps.	08771285

Supplementary Table A.6: Versions of external software used

A.4 Supplementary Notes

A.4.1 vg mpmap algorithm details

A.4.1.1 Seeding

Algorithm 1 contains pseudocode for the algorithm used to extract supermaximal exact match (SMEM) seeds. This algorithm utilizes a GCSA2 index to query suffix array intervals, which can be located in the graph similarly to the FM-index [65, 189]. After finding a longest maximal exact match (MEM), the algorithm navigates upward in the implicit suffix

tree using a longest common prefix (LCP) array. After computing the SMEMs of a read, we find the minimally-more-frequent MEMs for each SMEM, subject to a minimum length. This algorithm for detecting them (Algorithm 3) proceeds by querying the number of occurrences of a "probe substring" with a size equal to the minimum length (Algorithm 2). If the number of occurrences equal to the number of occurrences of the SMEM, then no minimally-more-frequent MEM that meets the minimum length criterion can contain that probe substring. Alternatively, if the substring occurs more frequently than the SMEM, then it must be part of a minimally-more-frequent MEM. The endpoint of this MEM is found using a bisection search. A benefit of the design of this algorithm is that, if the minimum length is longer than a match of a random sequence is likely to be, it is possible to skip over many characters in the SMEM without ever querying them.

A.4.1.2 Reachability between seeds

Algorithm 4 presents a simplified version of the algorithm for computing reachability between exact match seeds. The simplification lies in that seed positions are treated as synonymous with nodes in the graph. In reality, seeds correspond to a path through the graph, not single nodes. Also, collinear seeds sometimes overlap each other either on the read or in the graph. For instance, this occurs when there are indel errors in a homopolymer, in which case the MEMs on either side of the error will both try to match the entire homopolymer. The full algorithm in vg mpmap handles these nuances, but the details are cumbersome to present. The three stages of the reachability algorithm achieve linear run time in the typical case in different ways. In stage 1 (Algorithm 5), most transitive reachability relationships are not discovered, so

the number of edges is usually linear. In stage 2 (Algorithm 6), the quadratic factor is limited to the number of noncollinear seeds that can nonetheless reach each other in the pangenome graph. In stage 3 (Algorithm 7), an early stopping condition limits the quadratic factor to the number of seeds that have more than one collinear successor.

A.4.1.3 Dynamic programming with multiple traceback

The pseudocode in Algorithm 8 presents a simplified version of the multiple traceback algorithm in which only the two highest-scoring alignments are returned. The logic extends naturally to finding the top k alignments, but the details are somewhat complicated and not particularly enlightening. The pseudocode also presents the multiple-traceback algorithm for a Needleman-Wunsch alignment for the sake of simplicity, but the generalization to POA is straightforward. The insight underlying this algorithm is that the second highest-scoring traceback diverges from the highest-scoring one at the cell where the difference is minimized between the score that is entered in the dynamic programming matrix and the other scores that could have been entered there. Moreover, the score of this alignment differs from the highest-scoring alignment's score by exactly that difference.

Algorithm 1: Stage 1 of MEM finding

Input: GCSA2 index G, LCP array L, read R
Output: SMEMs between R and sequence graph

Function `FindSMEMs`(G, L, R)**:**
  $b \leftarrow |R|$, $e \leftarrow |R|$ `// ends of a match's read interval`
  $s \leftarrow G$.fullSAInterval() `// suffix array interval on the graph`
  while $b > 0$ **do**
    `// extend match by one character`
    $\hat{s} \leftarrow G$.LF$(s, R[b-1])$
    if $\hat{s}$.empty() **then**
      `// the SMEM is exhausted`
      yield (s, b, e)
      $n \leftarrow L$.parent(s) `// match's parent suffix tree node`
      $e \leftarrow b + n$.longestCommonPrefix()
      $s \leftarrow n$.interval()
    else
      `// the match was successful`
      $b \leftarrow b - 1$
      $s \leftarrow \hat{s}$
    end
  end
  yield (s, b, e)

Algorithm 2: Check if a substring is more frequent than an SMEM (Subroutine of Stage 2 of MEM-finding)

Input: GCSA2 index G, read R, SMEM (s, b, e), probe substring (pb, pe) with $pb \geq b$ and $pe \leq e$
Output: MEM (t, p, pe), where $p \geq pb$ is the minimum index such that (p, pe) occurs more times than (b, e) in the graph.

Function `MoreFrequentProbe`(G, R, (s, b, e), (pb, pe))**:**
  $p \leftarrow pe$
  $t \leftarrow G$.fullSAInterval()
  while $p > pb$ **do**
    $\hat{t} \leftarrow G$.LF$(t, R[p-1])$
    if G.count$(\hat{t}) > G$.count(s) **then**
      $p \leftarrow p - 1$
      $t \leftarrow \hat{t}$
    else
      break
    end
  end
  return (t, p, pe)

Algorithm 3: Stage 2 of MEM-finding

Input: GCSA2 index G, read R, SMEM (s,b,e), minimum length L
Output: All minimally-more-frequent MEMs

Function `MoreFrequentMEMs` (G, R, (s,b,e), L) **:**
  $mb \leftarrow b$, $me \leftarrow b+L$ `// the probe substring`
  while $me \leq e$ **do**
    $t, pb, pe \leftarrow$ `MoreFrequentProbe` (G, R, (s,b,e), (mb,me))
    if $pb \neq mb$ **then**
      `// probe cannot occur in any minimally-more-frequent MEM`
      $mb \leftarrow pb+1$
      $me \leftarrow mb+L$
    else
      `// probe occurs in minimally-more-frequent MEM, bisect to find end`
      $l \leftarrow me$, $h \leftarrow e$
      while $h \neq l$ **do**
        $c \leftarrow \lfloor (h+l)/2 \rfloor$
        $t, pb, pe \leftarrow$ `MoreFrequentProbe` (G, R, (s,b,e), (mb,c))
        if $pb \neq mb$ **then**
          $h \leftarrow c$
        else
          $l \leftarrow c$
        end
      end
      yield (t,mb,h)
      $me \leftarrow h+1$
      $mb \leftarrow me-L$
    end
  end

Algorithm 4: Construct the connectivity graph between seeds

Input: D a DAG, $S = \{(n, b, e)\}$ the MEM seeds (represented by a node n and a read interval $[b, e)$)

Output: Transitive reduction of graph with seeds as nodes and edges between the seeds if they are connected by a path in D and the read intervals are collinear.

Function `SeedGraph(`G, S`)`**:**

```
/* make an initial graph ignoring collinearity on the read
   */
```

$G_0 \leftarrow$ `TentativeSeedGraph(`D, S`)`

```
/* rewire edges to respect collinearity                    */
```

$G_1 \leftarrow$ `CollinearSeedGraph(`G_0`)`

```
/* remove transitive edges (if any)                        */
```

$G_2 \leftarrow$ `TransitiveReduction` (G_1)

return G_2

Algorithm 5: Stage 1 of constructing connectivity graph between seeds

Input: D a DAG, $S = \{(n, b, e)\}$, the set of MEM seeds represented by a node n from D, and a read interval $[b, e)$

Output: Graph with seeds for nodes and edges between two seeds whenever there is a walk in the graph that does not include any other seed

```
1  Function TentativeSeedGraph(G, S):
2      G_0.nodes() ← S                          // nodes correspond to seeds
3      foreach n in D.topologicalOrder() do
4          if ∃ some seed s = (n, b, e) then
               // seeds that can reach n can reach s
5              foreach t in n.predecessorSeeds() do
6                  G_0.addEdge(t, s)
7              end
               // s blocks earlier seeds and reaches n's successors
8              foreach m in n.successors() do
9                  m.predecessorSeeds().insert(s)
10             end
11         else
               // all seeds that reach n can reach its successors
12             foreach m in n.successors() do
13                 foreach s in n.predecessorSeeds() do
14                     m.predecessorSeeds().insert(s)
15                 end
16             end
17         end
18     end
19     return G_0
```

Algorithm 6: Stage 2 of constructing connectivity graph between seeds

Input: G_0, graph with seeds as nodes and edges indicating that there is a walk connecting the two seeds in the graph (may exclude transitive edges)

Output: Graph with seeds for nodes and edges indicating 1) that there is a walk connecting the two seeds, and 2) the seeds are collinear on the read. Transitive edges may be excluded.

```
1  Function CollinearSeedGraph(G_0):
2      G_1.nodes() ← G_0.nodes()
3      foreach s in G_0.topologicalOrder() do
           // initialize a queue with s's predecessors
4          q.init()
5          foreach t in s.predecessors() do
6              q.enqueue(t)
7          end
8          while q is not empty do
9              t = q.dequeue()
               // t is only in q if it can reach s, check for
                  collinearity
10             if t.readInterval() is collinear with s.readInterval() then
11                 G_1.addEdge(t, s)
                   // don't explore t's collinear predecessors, they
                      will only produce transitive edges
12             else
13                 s.noncollinearPredecessors().insert(t)
                   // must explore t's collinear predecessors
14                 foreach u in t.predecessors() do
15                     q.enqueue(u)
16                 end
17             end
               // always explore t's noncollinear predecessors
18             foreach u in t.noncollinearPredecessors() do
19                 q.enqueue(u)
20             end
21         end
22     end
23     return G_1
```

Algorithm 7: Stage 3 of constructing connectivity graph between seeds

Input: G_1, a DAG
Output: The transitive reduction of G_1

Function `TransitiveReduction (`G_1`)`**:**
  G_2.nodes() $\leftarrow$ G_1.nodes()
  foreach s **in** G_1.topologicalOrder() **do**
    if s.numSuccessors() $= 1$ **then**
      `// all walks out of` s `use the edge, it cannot be transitive`
      continue
    end
    `// keep track of which seeds have been visited`
    $v \leftarrow \varnothing$
    `// iterate over neighbors in topological order`
    foreach t **in** s.neighbors() **do**
      if $t \in v$ **then**
        `//` t `is reachable from an earlier edge,` (s,t) `is transitive`
        G_2.removeEdge(s,t)
      else
        `// mark all nodes reachable from this edge as visited`
        v.insert(t)
        foreach $u \in t$.reachableByDFS() **do**
          v.insert(u)
        end
      end
    end
  end

Algorithm 8: Find the two highest-scoring alignments from a single DP matrix

Input: M the dynamic programming matrix of an alignment, g gap penalty, S score matrix, Q_1 and Q_2 the sequences being aligned
Output: The two top-scoring alignments

Function `MultipleTraceback(`M,g,S,Q_1,Q_2`)`**:**
  $a_1, s_1, d, s_2 \leftarrow$ `OptimalTraceback(`M,g,S,Q_1,Q_2`)`
  $a_2 \leftarrow$ `NextBestTraceback(`M,g,S,Q_1,Q_2,d`)`
  return (a_1,s_1), (a_2,s_2)

Algorithm 9: Find the optimal traceback and the point of deflection for the second-best traceback

Input: M the dynamic programming matrix of an alignment, g gap penalty, S score matrix, Q_1 and Q_2 the sequences being aligned

Output: The optimal traceback and its score, and the next-best tracebacks score and point deflection from the optimal traceback

Function `OptimalTraceback`(M, g, S, Q_1, Q_2)**:**
- $i = M$.numRows(), $j = M$.numCols()
- $s \leftarrow M[i, j]$ `// the alignment score`
- $a \leftarrow \varnothing$ `// alignment to trace`
- $d \leftarrow \varnothing$ `// point of deflection for 2nd alignment`
- $\Delta \leftarrow \infty$ `// score difference for 2nd alignment`
- **while** $i \neq 0$ or $j \neq 0$ **do**
  - **if** $M[i, j] = M[i-1, j-1] + S[Q_1[i-1], Q_2[j-1]]$ **then**
    - $i', j' \leftarrow i-1, j-1$
  - **else if** $M[i, j] = M[i-1, j] - g$ **then**
    - $i', j' \leftarrow i-1, j$
  - **else**
    - $i', j' \leftarrow i, j-1$
  - **end**
  - a.prepend(i', j')
  - `// look at suboptimal extensions to find next-best traceback`
  - **if** $i', j' \neq i-1, j-1$ and $M[i, j] - (M[i-1, j-1] + S[Q_1[i-1], Q_2[j-1]]) < \Delta$ **then**
    - $\Delta \leftarrow M[i, j] - (M[i-1, j-1] + S[Q_1[i-1], Q_2[j-1]])$
    - $d \leftarrow (i, j \longrightarrow i-1, j-1)$
  - **end**
  - **if** $i', j' \neq i-1, j$ and $M[i, j] - (M[i-1, j] - g) < \Delta$ **then**
    - $\Delta \leftarrow M[i, j] - (M[i-1, j] - g)$
    - $d \leftarrow (i, j \longrightarrow i-1, j)$
  - **end**
  - **if** $i', j' \neq i, j-1$ and $M[i, j] - (M[i, j-1] - g) < \Delta$ **then**
    - $\Delta \leftarrow M[i, j] - (M[i, j-1] - g)$
    - $d \leftarrow (i, j \longrightarrow i, j-1)$
  - **end**
  - $i, j \leftarrow i', j'$
- **end**
- **return** $(a, s, d, s - \Delta)$

Algorithm 10: Find the second-highest scoring alignment

Input: M the dynamic programming matrix of an alignment, g gap penalty, S score matrix, Q_1 and Q_2 the sequences being aligned, d the point of deflection from the optimal traceback

Output: The second-highest scoring alignment

Function `NextBestTraceback`(M, g, S, Q_1, Q_2, d)**:**

- $i = M$.numRows(), $j = M$.numCols()
- $a \leftarrow \varnothing$ `// alignment to trace`
- **while** $i \neq 0$ or $j \neq 0$ **do**
  - **if** d.from() $= i, j$ **then**
    - `// this is where 2nd best traceback differs from optimal`
    - $i', j' \leftarrow d$.to()
  - **else if** $M[i, j] = M[i-1, j-1] + S[Q_1[i-1], Q_2[j-1]]$ **then**
    - $i', j' \leftarrow i-1, j-1$
  - **else if** $M[i, j] = M[i-1, j] - g$ **then**
    - $i', j' \leftarrow i-1, j$
  - **else**
    - $i', j' \leftarrow i, j-1$
  - **end**
  - a.prepend(i', j')
  - $i, j \leftarrow i', j'$
- **end**
- **return** a

Appendix B

Appendix B: Proofs of theorems regarding snarl compatibility

B.1 Appendix 1 and 2 of *Superbubbles, Ultrabubbles, and Cacti*

Omitted. I did not contribute to the proofs in these appendices.

B.2 Appendix 3 of *Supperbubbles, Ultrabubbles, and Cacti*

In this section, we prove Theorems 2 and 3, which characterize a sufficient condition to produce a family of compatible snarls. We begin with two useful lemmas.

Lemma 10 *Let $\{x, y\}$ be a snarl with snarl subgraph X. If u is a node in X and v is a node that is not in X, then any path from u to v includes the black edge incident on x or the black edge incident on y.*

Proof *Suppose a path exists that does not include either of the black edges incident on x and on y. Then u is not disconnected from v after deleting these edges, which contradicts the separability of* $\{x,y\}$.

Lemma 11 *Let* $\{x,y\}$ *be a snarl with subgraph X. Then there exists a path from u to either x or y that includes neither the black edge incident on x nor the black edge incident on y iff u is in X.*

Proof *First assume u is in X. Some path exists from u to either x or y, else u is not in the same connected component as x and y. Consider the shortest such path. Without loss of generality, assume this path is between u and x. Suppose the black edge incident on x or the black edge incident on y occurs somewhere along the path. Without loss of generality, assume it is the black edge incident on x. By Lemma 1, x or y must occur in the prefix of the path between u and* $\hat{x}$. *This implies that the path was not the shortest, which is a contradiction. Therefore, there exists a path from u that contains neither the black edge incident on x nor the black edge incident on y.*

Next assume without loss of generality that a path exists from u to x that includes neither the black edge incident on x nor the black edge incident on y. This path is preserved after removing these two edges. This implies that u is in the same connected component as x (and hence also y) in the resulting graph, so u is in X.

Let $\{x_1,y_1\}$ and $\{x_2,y_2\}$ be two snarls with snarl subgraphs X_1 and X_2 respectively. We will say that $\{x_1,y_1\}$ *splits* $\{x_2,y_2\}$ if either a) x_2 is in X_1 but y_2 is not in X_1, or b) y_2 is in X_1 but x_2 is not in X_1. This condition clearly violates compatibility.

Lemma 12 *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be snarls with snarl subgraphs X_1 and X_2. If $\{x_1, y_1\}$ splits $\{x_2, y_2\}$, then x_1 and y_1 are in X_2.*

Proof *We will proceed by showing that all other cases lead to contradictions. Without loss of generality, assume x_2 is in X_1 and y_2 is not in X_1.*

Case I: x_1 and y_1 are not in X_2

Consider the set of paths from x_2 to y_2 that do not pass through $\hat{y}_2$ or $\hat{x}_2$. This set is nonempty else X_2 is disconnected. By Lemma 10, any such path must include x_1 or y_1, which would imply that x_1 is in X_2 or y_1 is in X_2 respectively by Lemma 11. This violates the assumption of the case, so this case is contradictory.

Case II: x_1 is in X_2 and y_1 is not in X_2

Any path from x_1 to y_1 that does not include the black edges incident on x and y cannot include y_2, else y_2 is in X_1 by Lemma 11. Therefore, it must contain the black edge incident on x_2 by Lemma 10. Without loss of generality, this implies that $\{x_1, x_2\}$ and $\{\hat{x}_2, y_1\}$ are separable, which violates the minimality of $\{x_1, y_1\}$. Thus, this case is contradictory as well.'

Case III: y_1 is in X_2 and x_1 is not in X_2

Same as Case II.

This proves the lemma.

Lemma 13 *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be snarls with snarl subgraphs X_1 and X_2. If $\{x_1, y_1\}$ splits $\{x_2, y_2\}$, then X_1 contains a black bridge edge.*

Proof *Without loss of generality, assume x_2 is in X_1 and y_2 is not in X_1. Then there exists at least one path from $\hat{x}_2$ to either x_1 or y_1 else X_1 is disconnected. By Lemma 12, x_1 and y_1 are*

in X_2, so all such paths must include the black edge incident on x_2 or the black edge incident on y_2 by Lemma 10. Since y_2 is not in X_1, all paths from x_1 or y_1 to $\hat{x}_2$ in X_1 must include the black edge incident on x_2. Therefore, the black edge incident on x_2 is a bridge edge by Menger's Theorem.

There are also cases that violate compatibility without splitting a snarl. The following lemmas characterize these cases.

Lemma 14 *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be snarls with distinct boundaries in a connected graph $B(D)$ whose snarl subgraphs are X_1 and X_2. If x_1 and y_1 are in X_2, and x_2 and y_2 are in X_1, then $X_1 \cup X_2 = B(D)$.*

Proof *Let u be an arbitrary node that is not in X_1 be arbitrary. There exists at least one path from u to a node in X_1 else $B(D)$ is not connected. Let p_1 be the shortest such path. Clearly, no node from X_1 occurs in p_1 except at its terminus. In particular, p_1 does not contain either $\hat{x}_2$ or $\hat{y}_2$. By Lemma 10, p_1 includes x_1 or y_1, so one of these must be the terminal node. Without loss of generality, assume it is x_1. Since x_1 is in X_2, there also exists a path p_2 from $\hat{x}_1$ to either x_2 or y_2 that does not include $\hat{x}_2$ or $\hat{y}_2$ by Lemma 11. Note that $p_1 p_2$ is a path from u to either x_2 or y_2 that does not include the black edges incident on x_2 and y_2. Thus, u is in X_2 by Lemma 11. This implies $X_1 \cup X_2 = B(D)$.*

Lemma 15 *Let $\{x, y_1\}$ and $\{x, y_2\}$ be snarls with snarl subgraphs X_1 and X_2. If $y_1 \neq y_2$, then X_1 and X_2 both contain a black bridge edge.*

Proof *Suppose y_2 is not in X_1. Then all paths from y_2 to x must include the black edge incident on x or the black edge incident on y_1 by Lemma 10. There exists at least one path between x*

and y_2 in X_2, which cannot include the black edge incident on x. Therefore, all paths between y_2 and x must include the black edge incident on y_1. This implies without loss of generality that $\{x, y_2\}$ and $\{\hat{y}_2, y_1\}$ are separable, which violates the minimality of $\{x, y_1\}$. Thus, y_2 is in X_1. Note that the black edge incident on x is not in X_1. Therefore, removing the black edge incident on y_2 from X_1 disconnects x from $\hat{y}_2$ because of the separability of $\{x, y_2\}$. Thus, the black edge incident on y_2 is a bridge edge in X_1. Similarly, the black edge incident on y_1 is a bridge edge in X_2.

Finally, we establish the relationship between pairs of snarls that allow for compatibility.

Lemma 16 *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be snarls with snarl subgraphs X_1 and X_2. If both x_2 and y_2 are in X_1, and both x_1 and y_1 are not in X_2, then $X_2 \subset X_1$.*

Proof *Let u be an arbitrary node in X_2. There exists a path p_1 from u to x_1 or y_1 that consists of only nodes in X_2 else X_2 is not connected. In particular, $\hat{x}_1, \hat{y}_1 \notin p_1$ else x_1 or y_1 would be in X_2. There also exists a path p_2 from x_2 to x_1 that includes neither $\hat{x}_1$ nor $\hat{y}_1$ by Lemma 11. The path $p_1 p_2$ connects u to x_1 and includes neither $\hat{x}_1$ nor $\hat{y}_1$. Thus, u is in X_1 by Lemma 11.*

Lemma 17 *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be snarls with snarl subgraphs X_1 and X_2. If x_2 and y_2 are not in X_1, and x_1 and y_1 are not in X_2, then X_1 and X_2 are disjoint.*

Proof *Let u be an arbitrary node in X_1, and let p be any path from u to x_2 or y_2. By Lemma 10, p includes x_1 or y_1. Thus, by Lemma 10, p includes $\hat{x}_2$ or $\hat{y}_2$. Since p was chosen arbitrarily, this implies u is not in X_2 by Lemma 11. Therefore, X_1 and X_2 are disjoint.*

Taken together, these results yield the sufficient condition for compatibility that we set out to prove.

Theorem 2 *In a connected biedged graph with at least one black bridge edge, the family of snarls whose subgraphs have no black bridge edges is compatible.*

Proof *Let $\{x_1, y_1\}$ and $\{x_2, y_2\}$ be arbitrary snarls with snarl subgraphs X_1 and X_2 such that neither subgraph contains a black bridge edge. By Lemma 13, neither snarl splits the other. By Lemma 15, the two snarls cannot share a boundary node. Therefore, we also cannot have both x_1 and y_1 in X_2, and x_2 and y_2 in X_1 else either X_1 or X_2 must contain the graph's black bridge edge by Lemma 14. This leaves three cases:*

1. *x_1 and y_1 are in X_2, and x_2 and y_2 are not in X_1*
2. *x_1 and y_1 are not in X_2, and x_2 and y_2 are in X_1*
3. *x_1 and y_1 are not in X_2, and x_2 and y_2 are not in X_1*

In the first two cases, one subgraph is nested in the other by Lemma 16. In last case, the subgraphs are disjoint by Lemma 17. Therefore, $\{x_1, y_1\}$ and $\{x_2, y_2\}$ are compatible.

We will now move on to proving that ultrabubbles are included in the family of snarls with no black bridge edges.

Lemma 18 *Let u be a terminal of a black bridge edge whose removal separates a graph $B(D)$ into connected components B_1 and B_2 with u in B_1 and $\hat{u}$ in B_2. Then all snarls $\{x, y\}$ have either both x and y in B_1 or both x and y in B_2.*

Proof *Suppose without loss of generality that x is in B_1 and y is in B_2. All paths between x and y include the black edge incident on u. Therefore, $\{x,u\}$ and $\{\hat{u},y\}$ are separable. This contradicts the minimality of $\{x,y\}$.*

Theorem 3 *No ultrabubble contains a black bridge edge in its subgraph.*

Proof *Let $\{x,y\}$ be an ultrabubble with subgraph X. Suppose X contains a black bridge edge with terminals u and $\hat{u}$. Removing this edge separates X into connected components X_1 and X_2 with u in X_1 and $\hat{u}$ in X_2. By Lemma 18, we may assume without loss of generality that x and y are in X_1.*

Since $X_2 \subset X$, there are no cyclic walks in X_2. Moreover, there is at least one edge in X_2 else the black bridge edge is a tip. Let w be the longest biedged walk starting from $\hat{u}$ in X_2. This walk must exist, since walks of unbounded length could only exist if there is a cyclic biedged walk. Moreover, w is not empty since X_2 contains at least one edge.

Suppose the final edge in w is gray. Since x and y are not in X_2, one endpoint of this gray edge must have no black edge incident on it in the full graph else w could be lengthened. This violates the definition of a bidirected graph. Therefore the final edge in w must be black. However, this implies that one endpoint of this black edge has no gray edges incident on it, else w could be lengthened. That is, the black edge is a tip, which contradicts the definition of ultrabubble. Therefore, X does not contain a black bridge edge.

Lemma 19 *Let $\{x,y\}$ be a snarl with subgraph X. Further, let u be any node, and let p be the shortest path from u to either x or y and q the shortest path from u to either $\hat{x}$ or $\hat{y}$. If u is in X, then $|p| < |q|$, and if u is not in X then $|q| < |p|$.*

Proof *First assume that u is in X. Then q contains x or y by Lemma 10. The subpath up to this point is a path between u and either x or y that is strictly shorter than q. Therefore,* $|p| < |q|$. *Similarly, if u is in X then* $|q| < |p|$.

Lemma 20 *Let* $(u,\hat{u})$ *be a black bridge edge that separates a graph* $B(D)$ *into connected components* B_1 *and* B_2 *with u in* B_1 *and* $\hat{u}$ *in* B_2. *Further, let* $\{x,y\}$ *be a snarl subgraph X such that a) x is in* B_2 *and b)* $x,y \neq \hat{u}$. *Then X contains* $\hat{u}$ *iff X contains w.*

Proof *By Lemma 18, y is in* B_2 *as well as x. Note that if* $x = \hat{u}$ *or* $y = \hat{u}$ *then the claim is verified trivially, so we may focus on the case where* $x \neq \hat{u}$ *and* $y \neq \hat{u}$. *In this case, the black edges* $(x,\hat{x})$ *and* $(y,\hat{y})$ *must be in* B_2.

First, assume $\hat{u}$ *is in X. There exists a path* p_1 *from w to u in* B_1. *Note that this implies* p_1 *contains neither* $(x,\hat{x})$ *nor* $(y,\hat{y})$. *There also exists a path* p_2 *from* $\hat{u}$ *to x or y that includes neither* $(x,\hat{x})$ *nor* $(y,\hat{y})$ *by Lemma 11. Thus,* p_1p_2 *is a path from w to x or y that includes neither* $(x,\hat{x})$ *nor* $(y,\hat{y})$, *which implies w is in X by Lemma 11.*

Next, assume w is in X. There exists a path p from w to x or y that includes neither $(x,\hat{x})$ *nor* $(y,\hat{y})$ *by Lemma 11. Since x and y are in* B_2, *this path must include* $(u,\hat{u})$, *which means it includes a subpath from* $\hat{u}$ *to x or y. Therefore,* $\hat{u}$ *is in X by Lemma 11*

Lemma 21 *Let* $(x,\hat{x})$ *be a black bridge edge whose removal separates a graph* $B(D)$ *into connected components* B_1 *and* B_2 *with x in* B_1 *and* $\hat{x}$ *in* B_2. *If* $\{x,y\}$ *is a snarl with subgraph X, then* $X \subset B_1$

Proof *X consists of only nodes that can be reached from x without crossing* $(x,\hat{x})$ *by Lemma 11. Therefore,* $X \subset B_1$.

Bibliography

[1] Alexej Abyzov, Alexander E Urban, Michael Snyder, and Mark Gerstein. CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing. *Genome Research*, 21(6):974–984, 2011.

[2] Vitor R C Aguiar, Jônatas César, Olivier Delaneau, Emmanouil T Dermitzakis, and Diogo Meyer. Expression estimation and eQTL mapping for HLA genes with a personalized pipeline. *PLoS genetics*, 15(4):e1008091, April 2019.

[3] Cornelis A Albers, Gerton Lunter, Daniel G MacArthur, Gilean McVean, Willem H Ouwehand, and Richard Durbin. Dindel: Accurate indel calls from short-read data. *Genome Research*, 21(6):961–973, 2011.

[4] Max A Alekseyev and Pavel A Pevzner. Breakpoint graphs and ancestral genome reconstructions. 19(5):943–957, May 2009.

[5] Carlos Alonso-Blanco, Jorge Andrade, Claude Becker, Felix Bemm, Joy Bergelson, Karsten M Borgwardt, Jun Cao, Eunyoung Chae, Todd M Dezwaan, Wei Ding, et al. 1,135 genomes reveal the global pattern of polymorphism in Arabidopsis thaliana. *Cell*, 166(2):481–491, 2016.

[6] Adam Ameur, Johan Dahlberg, Pall Olason, Francesco Vezzi, Robert Karlsson, Marcel Martin, Johan Viklund, Andreas Kusalananda Kähäri, Pär Lundin, Huiwen Che, et al. SweGen: a whole-genome data resource of genetic variability in a cross-section of the Swedish population. *European Journal of Human Genetics*, 25(11):1253–1260, 2017.

[7] Jérôme Amilhastre, Marie-Catherine Vilarem, and Philippe Janssen. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete Applied Mathematics*, 86(2-3):125–144, 1998.

[8] Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. In *Lecture Notes in Computer Science*, pages 160–173. Springer Berlin Heidelberg, 1997.

[9] Dmitry Antipov, Anton Korobeynikov, Jeffrey S. McLean, and Pavel A. Pevzner. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32(7):1009–1015, November 2015.

[10] Joel Armstrong, Ian T Fiddes, Mark Diekhans, and Benedict Paten. Whole-genome alignment and comparative annotation. *Annual review of animal biosciences*, 7:41–64, 2019.

[11] Joel Armstrong, Glenn Hickey, Mark Diekhans, Ian T Fiddes, Adam M Novak, Alden Deran, Qi Fang, Duo Xie, Shaohong Feng, Josefin Stiller, et al. Progressive Cactus is a multiple-genome aligner for the thousand-genome era. *Nature*, 587(7833):246–251, 2020.

[12] Peter A Audano, Arvis Sulovari, Tina A Graves-Lindsay, Stuart Cantsilieris, Melanie Sorensen, AnneMarie E Welch, Max L Dougherty, Bradley J Nelson, Ankeeta Shah, Susan K Dutcher, et al. Characterizing the major structural variant alleles of the human genome. *Cell*, 176(3):663–675, 2019.

[13] Tomas Babak, Brian DeVeale, Emily K Tsang, Yiqi Zhou, Xin Li, Kevin S Smith, Kim R Kukurba, Rui Zhang, Jin Billy Li, Derek van der Kooy, et al. Genetic conflict reflected in tissue-specific maps of genomic imprinting in human and mouse. *Nature genetics*, 47(5):544–549, 2015.

[14] Haihua Bai, Xiaosen Guo, Narisu Narisu, Tianming Lan, Qizhu Wu, Yanping Xing, Yong Zhang, Stephen R Bond, Zhili Pei, Yanru Zhang, et al. Whole-genome sequencing of 175 Mongolians uncovers population-specific genetic architecture and gene flow throughout North and East Asia. *Nature genetics*, 50(12):1696–1704, 2018.

[15] Anton Bankevich, Andrey Bzikadze, Mikhail Kolmogorov, and Pavel A. Pevzner. Assembling long accurate reads using de Bruijn graphs. *bioRxiv*, page 2020.12.10.420448, December 2020. Publisher: Cold Spring Harbor Laboratory Section: New Results.

[16] Yael Baran, Meena Subramaniam, Anne Biton, Taru Tukiainen, Emily K Tsang, Manuel A Rivas, Matti Pirinen, Maria Gutierrez-Arcelus, Kevin S Smith, Kim R Kukurba, et al. The landscape of genomic imprinting across diverse adult human tissues. *Genome research*, 25(7):927–936, 2015.

[17] Lorenzo Barchi, Mark Timothy Rabanus-Wallace, Jaime Prohens, Laura Toppino, Sudharsan Padmarasu, Ezio Portis, Giuseppe Leonardo Rotino, Nils Stein, Sergio Lanteri, and Giovanni Giuliano. Improved genome assembly and pan-genome provide key insights on eggplant domestication and breeding. *The Plant Journal*, 2021.

[18] Anders Bergström, Shane A McCarthy, Ruoyun Hui, Mohamed A Almarri, Qasim Ayub, Petr Danecek, Yuan Chen, Sabine Felkel, Pille Hallast, Jack Kamm, et al. Insights into human genetic variation and population history from 929 diverse genomes. *Science*, 367(6484), 2020.

[19] Doruk Beyter, Helga Ingimundardottir, Asmundur Oddsson, Hannes P Eggertsson, Eythor Bjornsson, Hakon Jonsson, Bjarni A Atlason, Snaedis Kristmundsdottir, Svenja

Mehringer, Marteinn T Hardarson, et al. Long-read sequencing of 3,622 Icelanders provides insight into the role of structural variants in human diseases and other traits. *Nature Genetics*, pages 1–8, 2021.

[20] Etienne Birmelé, Pierluigi Crescenzi, Rui Ferreira, Roberto Grossi, Vincent Lacroix, Andrea Marino, Nadia Pisanti, Gustavo Sacomoto, and Marie-France Sagot. Efficient bubble enumeration in directed graphs. In Liliana Calderón-Benavides, Cristina González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *String Processing and Information Retrieval: 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, pages 118–129, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[21] Dorret I Boomsma, Cisca Wijmenga, Eline P Slagboom, Morris A Swertz, Lennart C Karssen, Abdel Abdellaoui, Kai Ye, Victor Guryev, Martijn Vermaat, Freerk Van Dijk, et al. The Genome of the Netherlands: design, and project goals. *European Journal of Human Genetics*, 22(2):221–227, 2014.

[22] Ljiljana Brankovic, Costas S Iliopoulos, Ritu Kundu, Manal Mohamed, Solon P Pissis, and Fatima Vayani. Linear-time superbubble identification algorithm for genome assembly. *Theoretical Computer Science*, pages –, May 2015.

[23] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature biotechnology*, 34(5):525–527, 2016.

[24] Wolfgang Brehm. Hash tables with pseudorandom global order. *INFOCOMP Journal of Computer Science*, 18(1):20–25, 2019.

[25] Moisés Burset, Igor A Seledtsov, and Victor V Solovyev. Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic acids research*, 28(21):4346–746, 2017.

[26] Stanislaw Bylka, Adam Idzik, and Zsolt Tuza. Maximum cuts: Improvements and local algorithmic analogues of the Edwards-Erdos inequality. *Discrete Mathematics*, 194(1-3):39–58, 1999.

[27] Marta Byrska-Bishop, Uday S Evani, Xuefang Zhao, Anna O Basile, Haley J Abel, Allison A Regier, André Corvelo, Wayne E Clarke, Rajeeva Musunuri, Kshithija Nagulapalli, et al. High coverage whole genome sequencing of the expanded 1000 Genomes Project cohort including 602 trios. *bioRxiv*, 2021.

[28] Stephane E Castel, Ami Levy-Moonshine, Pejman Mohammadi, Eric Banks, and Tuuli Lappalainen. Tools and best practices for data processing in allelic expression analysis. *Genome Biology*, 16(1), September 2015.

[29] Stephane E Castel, Pejman Mohammadi, Wendy K Chung, Yufeng Shen, and Tuuli Lappalainen. Rare variant phasing and haplotypic expression from RNA sequencing with phASER. *Nature communications*, 7(1):1–6, 2016.

[30] Mark JP Chaisson, John Huddleston, Megan Y Dennis, Peter H Sudmant, Maika Malig, Fereydoun Hormozdiari, Francesca Antonacci, Urvashi Surti, Richard Sandstrom, Matthew Boitano, et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature*, 517(7536):608–611, 2015.

[31] Mark JP Chaisson, Ashley D Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J Gardner, Oscar L Rodriguez, Li Guo, Ryan L Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature Communications*, 10(1):1784, 2019.

[32] Xian Chang, Jordan Eizenga, Adam M Novak, Jouni Sirén, and Benedict Paten. Distance indexing and seed clustering in sequence graphs. *Bioinformatics*, 36(Supplement 1):i146–i153, 2020.

[33] Nae-Chyun Chen, Brad Solomon, Taher Mun, Sheila Iyer, and Ben Langmead. Reference flow: reducing reference bias using multiple population genomes. *Genome biology*, 22(1):1–17, 2021.

[34] Charleston WK Chiang, Serghei Mangul, Christopher Robles, and Sriram Sankararaman. A comprehensive map of genetic variation in the world's largest ethnic group—Han Chinese. *Molecular biology and evolution*, 35(11):2736–2750, 2018.

[35] Deanna M Church, Valerie A Schneider, Tina Graves, Katherine Auger, Fiona Cunningham, Nathan Bouk, Hsiu-Chuan Chen, Richa Agarwala, William M McLaren, Graham RS Ritchie, et al. Modernizing reference genome assemblies. *PLoS Biology*, 9(7):e1001091, 2011.

[36] Rachel M Colquhoun, Michael B Hall, Leandro Lima, Leah W Roberts, Kerri M Malone, Martin Hunt, Brice Letcher, Jane Hawkey, Sophie George, Louise Pankhurst, et al. Nucleotide-resolution bacterial pan-genomics with reference graphs. *bioRxiv*, 2020.

[37] 1000 Genomes Project Consortium et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.

[38] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56, 2012.

[39] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[40] Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.

[41] GenomeAsia100K Consortium et al. The GenomeAsia 100K Project enables genetic discoveries across Asia. *Nature*, 576(7785):106, 2019.

[42] International HapMap Consortium et al. A second generation human haplotype map of over 3.1 million snps. *Nature*, 449(7164):851, 2007.

[43] International HapMap 3 Consortium et al. Integrating common and rare genetic variation in diverse human populations. *Nature*, 467(7311):52, 2010.

[44] The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, September 2012.

[45] UK10K Consortium et al. The UK10K project identifies rare variants in health and disease. *Nature*, 526(7571):82–90, 2015.

[46] Danang Crysnanto, Alexander S Leonard, Zih-Hua Fang, and Hubert Pausch. Novel functional sequences uncovered through a bovine multiassembly graph. *Proceedings of the National Academy of Sciences*, 118(20), 2021.

[47] Danang Crysnanto and Hubert Pausch. Bovine breed-specific augmented reference graphs facilitate accurate sequence read mapping and unbiased variant discovery. *Genome biology*, 21(1):1–27, 2020.

[48] Danang Crysnanto, Christine Wurmser, and Hubert Pausch. Accurate sequence variant genotyping in cattle using variation-aware genome graphs. *Genetics Selection Evolution*, 51(1):1–15, 2019.

[49] Carrie A Davis, Benjamin C Hitz, Cricket A Sloan, Esther T Chan, Jean M Davidson, Idan Gabdank, Jason A Hilton, Kriti Jain, Ulugbek K Baymuradov, Aditi K Narayanan, Kathrina C Onate, Keenan Graham, Stuart R Miyasato, Timothy R Dreszer, J Seth Strattan, Otto Jolanki, Forrest Y Tanaka, and J Michael Cherry. The encyclopedia of DNA elements (ENCODE): data portal update. *Nucleic Acids Research*, 46(D1):D794–D801, November 2017.

[50] Eric T Dawson and Richard Durbin. GFAKluge: A C++ library and command line utilities for the graphical fragment assembly formats. *Journal of Open Source Software*, 4(33), 2019.

[51] N G de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 1(49):758–764, 1946.

[52] Jacob F Degner, John C Marioni, Athma A Pai, Joseph K Pickrell, Everlyne Nkadori, Yoav Gilad, and Jonathan K Pritchard. Effect of read-mapping biases on detecting allele-specific expression from RNA-sequencing data. *Bioinformatics*, 25(24):3207–3212, 2009.

[53] Luca Denti, Raffaella Rizzi, Stefano Beretta, Gianluca Della Vedova, Marco Previtali, and Paola Bonizzoni. ASGAL: aligning RNA-seq data to a splicing graph to detect novel alternative splicing events. *BMC bioinformatics*, 19(1):1–21, 2018.

[54] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[55] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[56] Peter Eades, Xuemin Lin, and William F Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993.

[57] Peter Ebert, Peter A Audano, Qihui Zhu, Bernardo Rodriguez-Martin, David Porubsky, Marc Jan Bonder, Arvis Sulovari, Jana Ebler, Weichen Zhou, Rebecca Serra Mari, et al. Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science*, 2021.

[58] Jana Ebler, Wayne E Clarke, Tobias Rausch, Peter A Audano, Torsten Houwaart, Jan Korbel, Evan E Eichler, Michael C Zody, Alexander T Dilthey, and Tobias Marschall. Pangenome-based genome inference. *bioRxiv*, 2020.

[59] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021.

[60] Jack Edmonds and Ellis L Johnson. Matching: A well-solved class of integer linear programs. pages 27–30. Springer Berlin Heidelberg, Berlin, Heidelberg, January 1970.

[61] John Eid, Adrian Fehr, Jeremy Gray, Khai Luong, John Lyle, Geoff Otto, Paul Peluso, David Rank, Primo Baybayan, Brad Bettman, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[62] Jordan M Eizenga, Adam M Novak, Emily Kobayashi, Flavia Villani, Cecilia Cisar, Simon Heumos, Glenn Hickey, Vincenza Colonna, Benedict Paten, and Erik Garrison. Efficient dynamic variation graphs. *Bioinformatics*, 36(21):5139–5144, 2020.

[63] Jordan M Eizenga, Adam M Novak, Jonas A Sibbesen, Simon Heumos, Ali Ghaffaari, Glenn Hickey, Xian Chang, Josiah D Seaman, Robin Rounthwaite, Mikko Ebler, Jana Rautiainen, Shilpa Garg, Benedict Paten, Tobias Marschall, Jouni Sirén, and Erik Garrison. Pangenome graphs. *Annual Review of Genomics and Human Genetics*, 21:139–162, 2020.

[64] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 1–10, 2008.

[65] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, November 2000.

[66] Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al.

GENCODE reference annotation for the human and mouse genomes. *Nucleic acids research*, 47(D1):D766–D773, 2019.

[67] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-Runs bounded space. *Journal of the ACM*, 67(1):1–54, January 2020.

[68] Lei Gao, Itay Gonda, Honghe Sun, Qiyue Ma, Kan Bao, Denise M Tieman, Elizabeth A Burzynski-Chang, Tara L Fish, Kaitlin A Stromberg, Gavin L Sacks, et al. The tomato pan-genome uncovers new genes and a rare allele regulating fruit flavor. *Nature genetics*, 51(6):1044–1051, 2019.

[69] Shilpa Garg, Mikko Rautiainen, Adam M Novak, Erik Garrison, Richard Durbin, and Tobias Marschall. A graph-based approach to diploid genome assembly. *Bioinformatics*, 34(13):i105–i114, 2018.

[70] Erik Garrison. *Graphical pangenomics*. PhD thesis, University of Cambridge, 2019.

[71] Erik Garrison. ekg/gimbricate. https://github.com/ekg/gimbricate, October 2020.

[72] Erik Garrison. ekg/seqwish. https://github.com/ekg/seqwish, February 2021.

[73] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*, 2012.

[74] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology*, 36(9):875–879, 2018.

[75] Agnieszka A Golicz, Philipp E Bayer, Guy C Barker, Patrick P Edger, HyeRan Kim, Paula A Martinez, Chon Kit Kenneth Chan, Anita Severn-Ellis, W Richard McCombie, Isobel AP Parkin, et al. The pangenome of an agronomically important crop plant Brassica oleracea. *Nature communications*, 7(1):1–8, 2016.

[76] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[77] Osamu Gotoh. Modeling one thousand intron length distributions with fitild. *Bioinformatics*, 34(19):3258–3264, 2018.

[78] Catherine Grasso and Christopher Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20(10):1546–1556, 2004.

[79] Cristian Groza, Tony Kwan, Nicole Soranzo, Tomi Pastinen, and Guillaume Bourque. Personalized and graph genomes reveal missing signal in epigenomic data. *Genome Biology*, 21(1), May 2020.

[80] Ivar Grytten, Knut D Rand, Alexander J Nederbragt, Geir O Storvik, Ingrid K Glad, and Geir K Sandve. Graph Peak Caller: Calling ChIP-seq peaks on graph-based reference genomes. *PLoS computational biology*, 15(2):e1006731, 2019.

[81] Daniel F Gudbjartsson, Hannes Helgason, Sigurjon A Gudjonsson, Florian Zink, Asmundur Oddson, Arnaldur Gylfason, Soren Besenbacher, Gisli Magnusson, Bjarni V Halldorsson, Eirikur Hjartarson, et al. Large-scale whole-genome sequencing of the Icelandic population. *Nature Genetics*, 47(5):435–444, 2015.

[82] Deepti Gurdasani, Tommy Carstensen, Fasil Tekola-Ayele, Luca Pagani, Ioanna Tachmazidou, Konstantinos Hatzikotoulas, Savita Karthikeyan, Louise Iles, Martin O Pollard, Ananyo Choudhury, et al. The African genome variation project shapes medical genetics in Africa. *Nature*, 517(7534):327–332, 2015.

[83] Andrew P Han. Human pangenome reference consortium releases data from 30 genomes. *genomeweb.com*, March 2021. [Online. Retrieved June 14, 2021.].

[84] F Harary and G E Uhlenbeck. On the number of Husimi trees: I. *Proceedings of the National Academy of Sciences of the United States of America*, 39(4):315–322, April 1953.

[85] Bernhard Haubold and Thomas Wiehe. How repetitive are genomes? *BMC bioinformatics*, 7(1):1–10, 2006.

[86] Glenn Hickey, David Heller, Jean Monlong, Jonas A Sibbesen, Jouni Sirén, Jordan Eizenga, Eric T Dawson, Erik Garrison, Adam M Novak, and Benedict Paten. Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome biology*, 21(1):1–17, 2020.

[87] John Huddleston, Mark JP Chaisson, Karyn Meltz Steinberg, Wes Warren, Kendra Hoekzema, David Gordon, Tina A Graves-Lindsay, Katherine M Munson, Zev N Kronenberg, Laura Vives, et al. Discovery and genotyping of structural variation from long-read haploid genome sequence data. *Genome research*, 27(5):677–685, 2017.

[88] Costas S Iliopoulos, Ritu Kundu, Manal Mohamed, and Fatima Vayani. Popping superbubbles and discovering clumps: Recent developments in biological sequence analysis. pages 3–14. Springer International Publishing, Cham, 2016.

[89] International HapMap Consortium et al. A haplotype map of the human genome. *Nature*, 437(7063):1299, 2005.

[90] International Human Genome Sequencing Consortium et al. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931, 2004.

[91] Pesho Ivanov, Benjamin Bichsel, Harun Mustafa, André Kahles, Gunnar Rätsch, and Martin Vechev. Astarix: Fast and optimal sequence-to-graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 104–119. Springer, 2020.

[92] Bharati Jadhav, Ramin Monajemi, Kristina K Gagalova, Daniel Ho, Harmen HM Draisma, Mark A van de Wiel, Lude Franke, Bastiaan T Heijmans, Joyce van Meurs, Rick Jansen, et al. RNA-seq in 296 phased trios provides a high-resolution map of genomic imprinting. *BMC biology*, 17(1):1–20, 2019.

[93] Chirag Jain, Sanchit Misra, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Accelerating sequence alignment to graphs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2019.

[94] Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the complexity of sequence-to-graph alignment. *Journal of Computational Biology*, 27(4):640–654, 2020.

[95] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

[96] Christine Jandrasits, Piotr W Dabrowski, Stephan Fuchs, and Bernhard Y Renard. seq-seq-pan: Building a computational pan-genome data structure on whole genome alignment. *BMC genomics*, 19(1):1–12, 2018.

[97] Murukarthick Jayakodi, Sudharsan Padmarasu, Georg Haberer, Venkata Suresh Bonthala, Heidrun Gundlach, Cécile Monat, Thomas Lux, Nadia Kamal, Daniel Lang, Axel Himmelbach, et al. The barley pan-genome reveals the hidden legacy of mutation breeding. *Nature*, 588(7837):284–289, 2020.

[98] Samuel Karlin and Stephen F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Science*, 87(6):2264–2268, 1990.

[99] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[100] Vaddadi Naga Sai Kavya, Kshitij Tayal, Rajgopal Srinivasan, and Naveen Sivadasan. Sequence alignment on directed graphs. *Journal of Computational Biology*, 26(1):53–67, January 2019.

[101] Jerome Kelleher, Yan Wong, Anthony W Wohns, Chaimaa Fadil, Patrick K Albers, and Gil McVean. Inferring whole-genome histories in large population datasets. *Nature genetics*, 51(9):1330–1338, 2019.

[102] Daehwan Kim, Ben Langmead, and Steven L Salzberg. HISAT: a fast spliced aligner with low memory requirements. *Nature methods*, 12(4):357–360, 2015.

[103] Daehwan Kim, Joseph M Paggi, Chanhee Park, Christopher Bennett, and Steven L Salzberg. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature biotechnology*, 37(8):907–915, 2019.

[104] Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14(4):R36, 2013.

[105] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):1–9, 2004.

[106] Eric S Lander, Lauren M Linton, Bruce Birren, Chad Nusbaum, Michael C Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. 2001.

[107] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357, 2012.

[108] Tuuli Lappalainen, Michael Sammeth, Marc R Friedländer, Peter Ac 't Hoen, Jean Monlong, Manuel A Rivas, Mar Gonzalez-Porta, Natalja Kurbatova, Thasso Griebel, Pedro G Ferreira, et al. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501(7468):506–511, 2013.

[109] Charity W Law, Yunshun Chen, Wei Shi, and Gordon K Smyth. voom: Precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome biology*, 15(2):1–17, 2014.

[110] Ryan M Layer, Colby Chiang, Aaron R Quinlan, and Ira M Hall. LUMPY: a probabilistic framework for structural variant discovery. *Genome Biology*, 15(6):R84, 2014.

[111] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[112] Wanseon Lee, Katharine Plant, Peter Humburg, and Julian C Knight. AltHapAlignR: improved accuracy of RNA-seq analyses through the use of alternative haplotypes. *Bioinformatics*, 34(14):2401–2408, 2018.

[113] Michal Levy-Sakin, Steven Pastor, Yulia Mostovoy, Le Li, Alden KY Leung, Jennifer McCaffrey, Eleanor Young, Ernest T Lam, Alex R Hastie, Karen HY Wong, et al. Genome maps across 26 human populations reveal population-specific patterns of structural variation. *Nature communications*, 10(1):1–14, 2019.

[114] Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-seq data with or without a reference genome. *BMC bioinformatics*, 12(1):1–16, 2011.

[115] Bo Li, Victor Ruotti, Ron M Stewart, James A Thomson, and Colin N Dewey. RNA-seq gene expression estimation with read mapping uncertainty. *Bioinformatics*, 26(4):493–500, December 2009.

[116] Heng Li. seqtk. `https://github.com/lh3/seqtk`.

[117] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, November 2011.

[118] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013.

[119] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[120] Heng Li, Xiaowen Feng, and Chong Chu. The design and construction of reference pangenome graphs with minigraph. *Genome biology*, 21(1):1–19, 2020.

[121] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennel, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[122] Heng Li, Shaun Jackman, Eugene Myers, Giorgio Gonnella, Paul Melsted, Isaac Turner, Michael L. Heuer, Jakub Wilk, Ilia Minkin, Gustavo Glusman, Egor Shcherbin, Erik Garrison, Eric Dawson, Brice Letcher, Steve Huang, and Jerven Bolleman. GFA specification. `https://https://github.com/GFA-spec/GFA-spec`, 2013.

[123] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, 2008.

[124] Jianying Li, Daojun Yuan, Pengcheng Wang, Qiongqiong Wang, Mengling Sun, Zhenping Liu, Huan Si, Zhongping Xu, Yizan Ma, Boyang Zhang, et al. Cotton pan-genome retrieves the lost sequences and genes during domestication and selection. *Genome biology*, 22(1):1–26, 2021.

[125] Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233, 2003.

[126] Yucheng Liu, Huilong Du, Pengcheng Li, Yanting Shen, Hua Peng, Shulin Liu, Guo-An Zhou, Haikuan Zhang, Zhi Liu, Miao Shi, et al. Pan-genome of wild and cultivated soybeans. *Cell*, 182(1):162–176, 2020.

[127] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome biology*, 15(12):1–21, 2014.

[128] Tsung-Yu Tony Lu, Mark JP Chaisson, Human Genome Structural Variation Consortium, et al. Profiling variable-number tandem repeat variation across populations using repeat-pangenome graphs. *bioRxiv*, pages 2020–08, 2021.

[129] Swapan Mallick, Heng Li, Mark Lipson, Iain Mathieson, Melissa Gymrek, Fernando Racimo, Mengyao Zhao, Niru Chennagiri, Susanne Nordenfelt, Arti Tandon, et al. The Simons Genome Diversity Project: 300 genomes from 142 diverse populations. *Nature*, 538(7624):201–206, 2016.

[130] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[131] Buwani Manuweera, Joann Mudge, Indika Kahanda, Brendan Mumey, Thiruvarangan Ramaraj, and Alan Cleary. Pangenome-wide association studies with frequented regions. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 627–632, 2019.

[132] Lasse Maretty, Jacob Malte Jensen, Bent Petersen, Jonas Andreas Sibbesen, Siyang Liu, Palle Villesen, Laurits Skov, Kirstine Belling, Christian Theil Have, Jose MG Izarzugaza, et al. Sequencing and de novo assembly of 150 genomes from Denmark as a population reference. *Nature*, 548(7665):87, 2017.

[133] Rui Martiniano, Erik Garrison, Eppie R Jones, Andrea Manica, and Richard Durbin. Removing reference bias and improving indel calling in ancient DNA data analysis by mapping to a sequence variation graph. *Genome biology*, 21(1):1–18, 2020.

[134] Duccio Medini, Claudio Donati, Hervé Tettelin, Vega Masignani, and Rino Rappuoli. The microbial pan-genome. *Current opinion in genetics & development*, 15(6):589–594, 2005.

[135] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of computational biology : a journal of computational molecular cell biology*, 16(8):1101–1116, August 2009.

[136] Michael L Metzker. Sequencing technologies—the next generation. *Nature reviews genetics*, 11(1):31–46, 2010.

[137] Zong Miao, Marcus Alvarez, Päivi Pajukanta, and Arthur Ko. ASElux: an ultra-fast and accurate allelic reads counter. *Bioinformatics*, 34(8):1313–1320, 2018.

[138] Alexander S Mikheyev and Mandy MY Tin. A first look at the Oxford Nanopore MinION sequencer. *Molecular ecology resources*, 14(6):1097–1102, 2014.

[139] Ilia Minkin and Paul Medvedev. Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ. *Nature communications*, 11(1):1–11, 2020.

[140] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by RNA-seq. *Nature methods*, 5(7):621–628, 2008.

[141] Eugene Myers and Webb Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[142] Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

[143] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(Supplement 2):ii79–ii85, 2005.

[144] Masao Nagasaki, Jun Yasuda, Fumiki Katsuoka, Naoki Nariai, Kaname Kojima, Yosuke Kawai, Yumi Yamaguchi-Kabata, Junji Yokozawa, Inaho Danjoh, Sakae Saito, et al. Rare variant discovery by deep whole-genome sequencing of 1,070 Japanese individuals. *Nature Communications*, 6:8018, 2015.

[145] Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1-2):455–463, 2000.

[146] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[147] Marius Nicolae, Serghei Mangul, Ion I Măndoiu, and Alex Zelikovsky. Estimation of alternative splicing isoform frequencies from RNA-seq data. *Algorithms for Molecular Biology*, 6(1), April 2011.

[148] Hassan Nikaein. hnikaein/stark. https://github.com/hnikaein/stark, January 2021.

[149] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, et al. The complete sequence of a human genome. *bioRxiv*, 2021.

[150] Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Detecting superbubbles in assembly graphs. In *Algorithms in bioinformatics*, pages 338–348. Springer, Heidelberg, Berlin, Heidelberg, 2013.

[151] James Orlin et al. Contentment in graph theory: covering graphs with cliques. In *Indagationes Mathematicae (Proceedings)*, volume 80, pages 406–424. North-Holland, 1977.

[152] Luca Pagani, Daniel John Lawson, Evelyn Jagoda, Alexander Mörseburg, Anders Eriksson, Mario Mitt, Florian Clemente, Georgi Hudjashov, Michael DeGiorgio, Lauri Saag, et al. Genomic analyses inform on migration events during the peopling of Eurasia. *Nature*, 538(7624):238–242, 2016.

[153] Benedict Paten, Mark Diekhans, Dent Earl, John St John, Jian Ma, Bernard Suh, and David Haussler. Cactus graphs for genome comparisons. *Journal of computational biology : a journal of computational molecular cell biology*, 18(3):469–481, March 2011.

[154] Benedict Paten, Jordan M Eizenga, Yohei M Rosen, Adam M Novak, Erik Garrison, and Glenn Hickey. Superbubbles, ultrabubbles, and cacti. *Journal of Computational Biology*, 25(7):649–663, 2018.

[155] Benedict Paten, Adam M Novak, Jordan M Eizenga, and Erik Garrison. Genome graphs and the evolution of genome inference. *Genome research*, 27(5):665–676, 2017.

[156] Benedict Paten, Daniel R Zerbino, Glenn Hickey, and David Haussler. A unifying model of genome evolution under parsimony. *BMC bioinformatics*, 15(1):1–31, 2014.

[157] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4):417–419, 2017.

[158] Pavel Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, March 2000.

[159] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[160] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, et al. A universal SNP and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10):983–987, 2018.

[161] Ryan Poplin, Valentin Ruano-Rubio, Mark A DePristo, Tim J Fennell, Mauricio O Carneiro, Geraldine A Van der Auwera, David E Kling, Laura D Gauthier, Ami Levy-Moonshine, David Roazen, Khalid Shakir, Joel Thibault, Sheila Chandran, Chris Whelan, Monkol Lek, Stacey Gabriel, Mark J Daly, Ben Neale, Daniel G MacArthur, and Eric Banks. Scaling accurate genetic variant discovery to tens of thousands of samples. *bioRxiv*, page 10.1101/201178, November 2018.

[162] Nicola. Prezza. A framework of dynamic data structures for string processing. In *International Symposium on Experimental Algorithms*. Leibniz International Proceedings in Informatics (LIPIcs), 2017.

[163] Peng Qin, Hongwei Lu, Huilong Du, Hao Wang, Weilan Chen, Zhuo Chen, Qiang He, Shujun Ou, Hongyu Zhang, Xuanzhao Li, et al. Pan-genome analysis of 33 genetically diverse rice accessions reveals hidden genomic variations. *Cell*, 2021.

[164] Aaron R Quinlan, Royden A Clark, Svetlana Sokolova, Mitchell L Leibowitz, Yujun Zhang, Matthew E Hurles, Joshua C Mell, and Ira M Hall. Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome research*, 20(5):623–635, 2010.

[165] Aaron R Quinlan and Ira M Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[166] Narayanan Raghupathy, Kwangbom Choi, Matthew J Vincent, Glen L Beane, Keith S Sheppard, Steven C Munger, Ron Korstanje, Fernando Pardo-Manual de Villena, and

Gary A Churchill. Hierarchical analysis of RNA-seq reads improves the accuracy of allele-specific expression. *Bioinformatics*, 34(13):2177–2184, 2018.

[167] Goran Rakocevic, Vladimir Semenyuk, Wan-Ping Lee, James Spencer, John Browning, Ivan J Johnson, Vladan Arsenijevic, Jelena Nadj, Kaushik Ghose, Maria C Suciu, et al. Fast and accurate genomic analyses using genome graphs. *Nature genetics*, 51(2):354–362, 2019.

[168] Matthew D Rasmussen, Melissa J Hubisz, Ilan Gronau, and Adam Siepel. Genome-wide inference of ancestral recombination graphs. *PLoS Genetics*, 10(5):e1004342, 2014.

[169] Tobias Rausch, Thomas Zichner, Andreas Schlattl, Adrian M Stütz, Vladimir Benes, and Jan O Korbel. DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics*, 28(18):i333–i339, 2012.

[170] Mikko Rautiainen, Dilip A Durai, Ying Chen, Lixia Xin, Hwee Meng Low, Jonathan Göke, Tobias Marschall, and Marcel H Schulz. AERON: Transcript quantification and gene-fusion detection using long reads. *bioRxiv*, page 2020.01.27.921338, 2020.

[171] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19):3599–3607, 2019.

[172] Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in O(V+ mE) time. *bioRxiv*, page 216127, 2017.

[173] Mikko Rautiainen and Tobias Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome biology*, 21(1):1–28, 2020.

[174] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen R. F. Twigg, Andrew O. M. WGS500 Consortium, Wilkie, Gil McVean, and Gerton Lunter. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912–918, 2014.

[175] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[176] Mark D Robinson, Davis J McCarthy, and Gordon K Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, 2010.

[177] Yohei Rosen, Jordan Eizenga, and Benedict Paten. *Describing the Local Structure of Sequence Graphs*, pages 24–46. Springer International Publishing, Cham, 2017.

[178] Joel Rozowsky, Alexej Abyzov, Jing Wang, Pedro Alves, Debasish Raha, Arif Harmanci, Jing Leng, Robert Bjornson, Yong Kong, Naoki Kitabayashi, et al. AlleleSeq: analysis of allele-specific expression and binding in a network framework. *Molecular systems biology*, 7(1):522, 2011.

[179] Frederick Sanger, Steven Nicklen, and Alan R Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977.

[180] Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10(9):1, 2009.

[181] Valerie A Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, Paul A Kitts, Terence D Murphy, Kim D Pruitt, Françoise Thibaud-Nissen, Derek Albracht, et al. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome research*, 27(5):849–864, 2017.

[182] Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt Von Haeseler, and Michael C Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature methods*, 15(6):461–468, 2018.

[183] Jeong-Sun Seo, Arang Rhie, Junsoo Kim, Sangjin Lee, Min-Hwan Sohn, Chang-Uk Kim, Alex Hastie, Han Cao, Ji-Young Yun, Jihye Kim, et al. De novo assembly and phasing of a Korean human genome. *Nature*, 538(7624):243–247, 2016.

[184] Kishwar Shafin, Trevor Pesout, Pi-Chuan Chang, Maria Nattestad, Alexey Kolesnikov, Sidharth Goel, Gunjan Baid, Jordan M Eizenga, Karen H Miga, Paolo Carnevali, et al. Haplotype-aware variant calling enables high accuracy in nanopore long-reads using deep neural networks. *bioRxiv*, 2021.

[185] Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E. Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, Fritz J. Sedlazeck, Tobias Marschall, Simon Mayes, Vania Costa, Justin M. Zook, Kelvin J. Liu, Duncan Kilburn, Melanie Sorensen, Katy M. Munson, Mitchell R. Vollger, Jean Monlong, Erik Garrison, Evan E. Eichler, Sofie Salama, David Haussler, Richard E. Green, Mark Akeson, Adam Phillippy, Karen H. Miga, Paolo Carnevali, Miten Jain, and Benedict Paten. Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nature Biotechnology*, 38(9):1044–1053, September 2020. Number: 9 Publisher: Nature Publishing Group.

[186] Rachel M Sherman, Juliet Forman, Valentin Antonescu, Daniela Puiu, Michelle Daya, Nicholas Rafaels, Meher Preethi Boorgula, Sameer Chavan, Candelaria Vergara, Victor E Ortega, et al. Assembly of a pan-genome from deep sequencing of 910 humans of african descent. *Nature genetics*, 51(1):30–35, 2019.

[187] Jonas A Sibbesen, Jordan M Eizenga, Adam M Novak, Jouni Sirén, Xian Chang, Erik Garrison, and Benedict Paten. Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. *bioRxiv*, 2021.

[188] Jonas Andreas Sibbesen, Lasse Maretty, and Anders Krogh. Accurate genotyping across variant classes and lengths using variant graphs. *Nature genetics*, 50(7):1054–1059, 2018.

[189] Jouni Sirén. Indexing variation graphs. In *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 13–27. SIAM, 2017.

[190] Jouni Sirén, Erik Garrison, Adam M Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, 2020.

[191] Jouni Sirén, Jean Monlong, Xian Chang, Adam M Novak, Jordan M Eizenga, Charles Markello, Jonas A Sibbesen, Glenn Hickey, Pi-Chuan Chang, Andrew Carroll, Namrata Gupta, Stacey Gabriel, Thomas W Blackwell, Aakrosh Ratan, Kent D Taylor, Stephen S Rich, Jerome I Rotter, David Haussler, Erik Garrison, and Benedict Paten. Genotyping common, large structural variations in 5,202 genomes using pangenomes, the Giraffe mapper, and the vg toolkit. *bioRxiv*, page 2020.12.04.412486, December 2021.

[192] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

[193] Lloyd M Smith, Jane Z Sanders, Robert J Kaiser, Peter Hughes, Chris Dodd, Charles R Connell, Cheryl Heiner, Stephen BH Kent, and Leroy E Hood. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321(6071):674–679, 1986.

[194] Temple F Smith and Michael S Waterman. Comparison of biosequences. *Advances in Applied Mathematics*, 2(4):482–489, 1981.

[195] Jia-Ming Song, Zhilin Guan, Jianlin Hu, Chaocheng Guo, Zhiquan Yang, Shuo Wang, Dongxu Liu, Bo Wang, Shaoping Lu, Run Zhou, et al. Eight high-quality genomes reveal pan-genome architecture and ecotype differentiation of Brassica napus. *Nature Plants*, 6(1):34–45, 2020.

[196] David J States, Warren Gish, and Stephen F Altschul. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods*, 3(1):66–70, 1991.

[197] Shayna Stein, Zhi-xiang Lu, Emad Bahrami-Samani, Juw Won Park, and Yi Xing. Discover hidden splicing variations by mapping personal transcriptomes to personal genomes. *Nucleic acids research*, 43(22):10612–10622, 2015.

[198] Kraig R Stevenson, Joseph D Coolon, and Patricia J Wittkopp. Sources of bias in measures of allele-specific expression derived from RNA-seq data aligned to a single reference genome. *BMC Genomics*, 14(1):536, 2013.

[199] Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, et al. An

integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.

[200] Chen Sun, Zhiqiang Hu, Tianqing Zheng, Kuangchen Lu, Yue Zhao, Wensheng Wang, Jianxin Shi, Chunchao Wang, Jinyuan Lu, Dabing Zhang, et al. RPAN: rice pan-genome browser for 3000 rice genomes. *Nucleic acids research*, 45(2):597–605, 2017.

[201] Wing-Kin Sung, Kunihiko Sadakane, Tetsuo Shibuya, Abha Belorkar, and Iana Pyrogova. An $o(mlogm)$-time algorithm for detecting superbubbles. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 12(4):770–777, January 2015.

[202] Sonia Tarazona, Fernando García-Alcalde, Joaquín Dopazo, Alberto Ferrer, and Ana Conesa. Differential expression in RNA-seq: a matter of depth. *Genome research*, 21(12):2213–2223, 2011.

[203] Amalio Telenti, Levi CT Pierce, William H Biggs, Julia di Iulio, Emily HM Wong, Martin M Fabani, Ewen F Kirkness, Ahmed Moustafa, Naisha Shah, Chao Xie, et al. Deep sequencing of 10,000 human genomes. *Proceedings of the National Academy of Sciences*, 113(42):11901–11906, 2016.

[204] Hagen Tilgner, Fabian Grubert, Donald Sharon, and Michael P Snyder. Defining a personal, allele-specific, and single-molecule long-read transcriptome. *Proceedings of the National Academy of Sciences*, 111(27):9869–9874, June 2014.

[205] Manuel Tognon, Vincenzo Bonnici, Erik Garrison, Rosalba Giugno, and Luca Pinello. GRAFIMO: variant and haplotype aware motif scanning on pangenome graphs. *bioRxiv*, 2021.

[206] Valter Tucci, Anthony R Isles, Gavin Kelsey, Anne C Ferguson-Smith, Marisa S Bartolomei, Nissim Benvenisty, Déborah Bourc'his, Marika Charalambous, Catherine Dulac, Robert Feil, et al. Genomic imprinting and physiological processes in mammals. *Cell*, 176(5):952–965, 2019.

[207] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[208] Kavya Vaddadi, Rajgopal Srinivasan, and Naveen Sivadasan. Read mapping on genome variation graphs. In *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[209] Bryce Van De Geijn, Graham McVicker, Yoav Gilad, and Jonathan K Pritchard. WASP: allele-specific software for robust molecular quantitative trait locus discovery. *Nature methods*, 12(11):1061–1063, 2015.

[210] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*, 27(5):737–746, 2017.

[211] Günter P Wagner, Koryu Kin, and Vincent J Lynch. Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. *Theory in biosciences*, 131(4):281–285, 2012.

[212] Jeremiah Wala and Rameen Beroukhim. SeqLib: a c ++ API for rapid BAM manipulation, sequence alignment and sequence assembly. *Bioinformatics*, page btw741, December 2016.

[213] Sean Walkowiak, Liangliang Gao, Cecile Monat, Georg Haberer, Mulualem T Kassa, Jemima Brinton, Ricardo H Ramirez-Gonzalez, Markus C Kolodziej, Emily Delorean, Dinushika Thambugala, et al. Multiple wheat genomes reveal global variation in modern breeding. *Nature*, pages 1–7, 2020.

[214] Aaron M Wenger, Paul Peluso, William J Rowell, Pi-Chuan Chang, Richard J Hall, Gregory T Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D Olson, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology*, 37(10):1155–1162, 2019.

[215] Thomas D Wu, Jens Reeder, Michael Lawrence, Gabe Becker, and Matthew J Brauer. GMAP and GSNAP for genomic sequence alignment: enhancements to speed, accuracy, and functionality. In *Statistical genomics*, pages 283–334. Springer, 2016.

[216] Zhikun Wu, Zehang Jiang, Tong Li, Chuanbo Xie, Liansheng Zhao, Shuai Ouyang, Yizhi Liu, Tao Li, Zhi Xie, et al. Structural variants in Chinese population and their impact on phenotypes, diseases and population adaptation. *bioRxiv*, 2021.

[217] Dana Wyman, Gabriela Balderrama-Gutierrez, Fairlie Reese, Shan Jiang, Sorena Rahmanian, Stefania Forner, Dina Matheos, Weihua Zeng, Brian Williams, Diane Trout, Whitney England, Shu-Hui Chu, Robert C Spitale, Andrea J Tenner, Barbara J Wold, and Ali Mortazavi. A technology-agnostic long-read analysis pipeline for transcriptome discovery and quantification. *bioRxiv*, page 10.1101/672931, June 2020.

[218] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, May 2008.

[219] Yong Zhou, Dmytro Chebotarov, Dave Kudrna, Victor Llaca, Seunghee Lee, Shanmugam Rajasekar, Nahed Mohammed, Noor Al-Bader, Chandler Sobel-Sorenson, Praveena Parakkal, et al. A platinum standard pan-genome resource that represents the population structure of Asian rice. *Scientific data*, 7(1):1–11, 2020.

[220] Florian Zink, Droplaug N Magnusdottir, Olafur T Magnusson, Nicolas J Walker, Tiffany J Morris, Asgeir Sigurdsson, Gisli H Halldorsson, Sigurjon A Gudjonsson, Pall Melsted, Helga Ingimundardottir, Snædis Kristmundsdottir, Kristjan F Alexandersson, Anna Helgadottir, Julius Gudmundsson, Thorunn Rafnar, Ingileif Jonsdottir, Hilma Holm, Gudmundur I Eyjolfsson, Olof Sigurdardottir, Isleifur Olafsson, Gisli Masson,

Daniel F Gudbjartsson, Unnur Thorsteinsdottir, Bjarni V Halldorsson, Simon N Stacey, and Kari Stefansson. Insights into imprinting from parent-of-origin phased methylomes and transcriptomes. *Nature Genetics*, 50(11):1542–1552, October 2018.