

UNIVERSITY OF CALIFORNIA  
SANTA CRUZ

**DATA STRUCTURES AND ALGORITHMS FOR READ MAPPING TO  
PANGENOME GRAPHS**

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

DOCTOR OF PHILOSOPHY

in

BIOMOLECULAR ENGINEERING & BIOINFORMATICS

by

**Xian H. Chang**

December 2024

The Dissertation of Xian H. Chang  
is approved:

---

Professor Benedict Paten, Chair

---

Professor David Haussler

---

Professor Russ Corbett-Detig

---

Professor Jian Ma

---

Dean Peter Biehl  
Vice Provost and Dean of Graduate Studies

Copyright © by

Xian H. Chang

2024

# Table of Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>ix</b>
<b>Abstract</b>	<b>xiii</b>
<b>Dedication</b>	<b>xiv</b>
<b>Acknowledgments</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>2</b>
2.1 The human reference genome . . . . .	3
2.2 Pangenomics . . . . .	3
2.3 Variation graphs . . . . .	3
2.4 Snarl decomposition . . . . .	3
2.5 Read mapping . . . . .	3
2.5.1 Seeding . . . . .	3
2.5.2 Clustering . . . . .	3
2.5.3 Chaining . . . . .	3
2.5.4 Alignment . . . . .	3
2.5.5 Linear mappers . . . . .	3
2.5.6 Graph mappers . . . . .	3
<b>3 Distance Indexing</b>	<b>4</b>
3.1 Preface . . . . .	4
3.2 Introduction . . . . .	5
3.3 Background . . . . .	6
3.3.1 Sequence Graph Structure . . . . .	6
3.3.2 Snarl Decomposition . . . . .	7
3.3.3 Prior Research . . . . .	10

3.4	Minimum Distance . . . . .	11
3.4.1	Minimum Distance Index . . . . .	12
3.4.2	Minimum Distance Algorithm . . . . .	15
3.5	Clustering . . . . .	19
3.5.1	Problem . . . . .	19
3.5.2	Algorithm . . . . .	20
3.6	Methods and Results . . . . .	25
3.7	Conclusion . . . . .	30
3.8	Funding . . . . .	31
<b>4</b>	<b>Short read mapping</b>	<b>32</b>
4.1	Preface . . . . .	32
4.2	Structured abstract . . . . .	33
4.2.1	Introduction . . . . .	33
4.2.2	Rationale . . . . .	33
4.2.3	Results . . . . .	34
4.2.4	Conclusion . . . . .	35
4.3	Abstract . . . . .	35
4.4	Introduction . . . . .	37
4.5	Results . . . . .	38
4.5.1	Giraffe: Fast, haplotype-aware pangenome mapping . . . . .	38
4.5.2	Pangenome references for evaluation . . . . .	40
4.5.3	Giraffe and VG-MAP map accurately to human pangenomes . . . . .	40
4.5.4	Haplotype sampling improves read mapping . . . . .	42
4.5.5	Giraffe improves pangenome mapping speed . . . . .	44
4.5.6	Giraffe reduces allele mapping bias . . . . .	46
4.5.7	Giraffe genotyping outperforms best practices . . . . .	46
4.5.8	Giraffe generalizes beyond human . . . . .	48
4.5.9	Genotyping the SVs of the 5202 samples . . . . .	49
4.5.10	Diverse, clustered SVs . . . . .	51
4.5.11	Comprehensive SV frequency estimates . . . . .	54
4.5.12	Fine-tuning SVs with frequencies . . . . .	55
4.5.13	SV frequency population signatures . . . . .	55
4.5.14	SVs, genes, and expression . . . . .	57
4.6	Discussion . . . . .	60
4.7	Methods summary . . . . .	62
4.7.1	Evaluation . . . . .	62
4.8	Acknowledgments . . . . .	67
4.8.1	Funding . . . . .	67
4.8.2	Author contributions . . . . .	68
4.8.3	Competing interests . . . . .	69
4.9	Supplementary materials and methods . . . . .	71
4.9.1	Giraffe Algorithm . . . . .	71

4.9.2	Mapping evaluation . . . . .	90
4.9.3	Evaluation of reference allele bias . . . . .	98
4.9.4	Genotyping and evaluating short variants with Dragen . . . . .	98
4.9.5	Genotyping and evaluating structural variants . . . . .	103
4.9.6	Combining long-read catalogs for an SV pangenome . . . . .	103
4.9.7	Population scale analysis of SVs . . . . .	105
4.9.8	Supplementary figures . . . . .	114
4.9.9	Supplementary tables . . . . .	122
<b>5</b>	<b>Long read mapping</b>	<b>156</b>
5.1	Preface . . . . .	156
<b>6</b>	<b>Discussion</b>	<b>157</b>
<b>A</b>	<b>Additional short read giraffe materials</b>	<b>158</b>
A.1	Preface . . . . .	158
	<b>Bibliography</b>	<b>159</b>

# List of Figures

3.1	Snarl tree example	7
3.2	Minimum distance calculation	12
3.3	Chain distances	14
3.4	Cyclic chain	14
3.5	Distance to ends of parent calculation	18
3.6	Clustering algorithm	22
3.7	Run times for distance algorithms	27
3.8	Distance calculations on a graph with simulated structural variants	29
3.9	Run time growth of clustering algorithm	30
4.1	Overview of short read giraffe experiments	36
4.2	Haplotype mapping	41
4.3	Simulated read mapping	43
4.4	Runtime and memory usage	45
4.5	Evaluating Giraffe for genotyping	47
4.6	SVs in the MESA cohort	52

4.7	Population-specific SVs and SV-eQTLs in the 1000 Genomes Project dataset . . . . .	58
4.8	Sequence graph example . . . . .	114
4.9	Diagram of minimizers, agglomerations, and regions . . . . .	115
4.10	Simulated read mapping with NovaSeq 6000 reads . . . . .	116
4.11	Simulated read mapping with HiSeq X Ten reads . . . . .	117
4.12	Simulated read mapping with HiSeq 2500 reads . . . . .	118
4.13	QQ plot for simulated read mapping with NovaSeq 6000 reads . . . . .	119
4.14	QQ plot for simulated read mapping with HiSeq X Ten reads . . . . .	120
4.15	QQ plot for simulated read mapping with HiSeq 2500 reads . . . . .	121
4.16	Assessing haplotype sampling and path cover . . . . .	122
4.17	Genotyping evaluation for the Dragen genotyper with projected mappings from Giraffe and other mappers . . . . .	123
4.18	Genotyping evaluation for the Dragen genotyper with projected mappings from Giraffe and other mappers in 1000GP regions only . . . . .	124
4.19	Genotyping evaluation for the Dragen genotyper with projected mappings from Giraffe and other mappers in 1000GP excluded regions . . . . .	125
4.20	Genotyping evaluation for the DeepVariant genotyper with projected mappings from Giraffe and other mappers . . . . .	126
4.21	SV evaluation of vg and GraphTyper on HGSVC and GIAB . . . . .	127
4.22	Mapping speed on yeast data . . . . .	128
4.23	Mapping speed on human data . . . . .	129
4.24	Structural variant genotyping workflow . . . . .	130

4.25 Minimal effect of down-sampling reads to 20x depth on the SV genotyping performance . . . . .	131
4.26 SV alleles and SV sites in the MESA cohort . . . . .	140
4.27 Four examples of SV sites in the MESA cohort . . . . .	141
4.28 Structural variants in 2,504 unrelated individuals from the 1000 Genomes Project	142
4.29 SV evaluation using 602 trios from the 1000 Genomes Project . . . . .	143
4.30 Allele frequency of SVs in this study and public studies . . . . .	144
4.31 Principal component analysis using the SV genotypes or SNVs in the MESA dataset . . . . .	145
4.32 SV sites with population specific frequency patterns in the MESA cohort . . . . .	146
4.33 PCA of SV genotypes in the 1000 Genomes Project . . . . .	147
4.34 SV sites with population specific frequency patterns in the 2,504 individuals from the 1000 Genomes Project . . . . .	148
4.35 P-value distribution of the SV vs gene expression tests in the GEUVADIS dataset	149
4.36 Location of the SV-eQTLs around the associated gene . . . . .	150
4.37 Enrichment of SV-eQTLs in highly expressed genes . . . . .	151
4.38 Example of a SV lead-eQTL . . . . .	151

# List of Tables

3.1	Primitive functions for the minimum distance algorithm . . . . .	17
4.1	Mapping accuracy on NovaSeq 6000 reads mapped to the 1000GP graph/GRCh38 reference . . . . .	132
4.2	Mapping accuracy on HiSeq X Ten reads mapped to the 1000GP graph/GRCh38 reference . . . . .	133
4.3	Mapping accuracy on HiSeq 2500 reads mapped to the 1000GP graph/GRCh38 reference . . . . .	134
4.4	Mapping accuracy on NovaSeq 6000 reads mapped to the HGSVC graph/GRCh38 reference . . . . .	135
4.5	Mapping accuracy on HiSeq X Ten reads mapped to the HGSVC graph/GRCh38 reference . . . . .	136
4.6	Mapping accuracy on HiSeq 2500 reads mapped to the HGSVC graph/GRCh38 reference . . . . .	137
4.7	Mapping accuracy using path cover GBWT . . . . .	138
4.8	Mapping accuracy using haplotype sampled GBWT . . . . .	139

4.9 Genotyping evaluation with VCFeval in HG002 using 150bp paired-end reads against the grch38-based graph reference . . . . .	139
4.10 Genotyping evaluation with Hap.py in HG002 using 150bp paired-end reads against the grch38-based graph reference . . . . .	140
4.11 Genotyping evaluation with VCFeval in HG002-exclusive variants using 250bp paired reads against the grch38-based graph reference . . . . .	145
4.12 Genotyping evaluation with Hap.py in HG002 using 250bp paired reads against the grch38-based graph reference . . . . .	147
4.13 Genotyping evaluation with VCFeval in HG002 using 150bp paired-end reads against the grch38-based graph reference in 1000GP variant regions . . . . .	149
4.14 Genotyping evaluation with VCFeval in HG002 using 150bp paired-end reads against the grch38-based graph reference in regions excluding the 1000GP variant set . . . . .	150
4.15 Genotyping evaluation between DeepVariant calls of bwamem and giraffe 1000GP alignments against grch38-based references with RTG VCFeval in HG003 using 35x 150bp paired-end reads . . . . .	152
4.16 Genotyping evaluation between DeepVariant calls of bwamem and giraffe 1000GP alignments against grch38-based references with Hap.py in HG003 using 35x 150bp paired-end reads . . . . .	152
4.17 Computing resources used when genotyping 2,000 samples from the MESA cohort and 3,202 samples from the 1000 Genomes Project dataset . . . . .	153
4.18 Most frequent Mendelian errors for SVs in the 1000GP trios . . . . .	153

4.19	Comparing SVs genotyped in the two datasets with two public SV catalogs . . .	154
4.20	Gene families enriched in SV-eQTLs and eGenes with only SV-eQTLs . . . . .	154
4.21	Computing resources for each task in the WDL genotyping workflow . . . . .	154
4.22	Table of parameters for GRCh38-based genotyping experiment vg runs . . . . .	155

# List of Algorithms

1	Algorithm for finding the distance from a position to the bounds of an ancestor snarl or chain . . . . .	18
2	Algorithm for finding the minimum distance between two positions in a variation graph . . . . .	19
3	Algorithm for finding clusters of positions on a snarl . . . . .	23
4	Algorithm for finding clusters of positions on a chain . . . . .	24
5	Algorithm for finding clusters of positions on a graph . . . . .	25

## **Abstract**

Data structures and algorithms for read mapping to pangenome graphs

by

Xian H. Chang

Theses have elements. Isn't that nice?

To myself,

Perry H. Disdainful,

the only person worthy of my company.

## **Acknowledgments**

I want to “thank” my committee, without whose ridiculous demands, I would have graduated so, so, very much faster.

# **Chapter 1**

## **Introduction**

Every dissertation should have an introduction. You might not realize it, but the introduction should introduce the concepts, background, and goals of the dissertation.

## **Chapter 2**

# **Background**

## **2.1 The human reference genome**

## **2.2 Pangenomics**

## **2.3 Variation graphs**

## **2.4 Snarl decomposition**

## **2.5 Read mapping**

### **2.5.1 Seeding**

### **2.5.2 Clustering**

### **2.5.3 Chaining**

### **2.5.4 Alignment**

### **2.5.5 Linear mappers**

### **2.5.6 Graph mappers**

# **Chapter 3**

## **Distance Indexing**

### **3.1 Preface**

This chapter is the full text of my paper "Distance Indexing and Seed Clustering in Sequence Graphs" [16], which I presented at ISMB in 2020 and was published in Bioinformatics. I am the sole first author of this paper. Jordan M. Eizenga originally conceptualized the idea of a snarl decomposition-based distance index and provided guidance and support throughout the process of software development and writing. Jordan M. Eizenga, Adam M. Novak, and Jouni Sirén provided minor contributions to the software development and conceptualization.

While the majority of my work is done for variation graphs, the distance index is generalizable to sequence graphs as well. Some notation in this chapter may differ from the rest of the thesis but the underlying structures they describe are the same.

## 3.2 Introduction

Conventional reference genomes represent genomes as a string or collection of strings. Accordingly, these so-called ‘linear reference genomes’ can only store one allele at each locus. The resulting lack of diversity introduces a systematic bias that makes samples look more like the reference genome [93]. This reference bias can be reduced by using pangenomic models, which incorporate the genomic content of populations of individuals [22]. Sequence graphs are a popular representation of pangenomes that can express all of the variation in a pangenome [64]. Sequence graphs have a more complex structure and the potential to contain more data than linear genomes. This tends to make functions on a sequence graph more computationally challenging than analogous functions on linear genomes.

One such function is computing distance. In a linear genome, the exact distance between two loci can be found by simply subtracting the offset of one locus from the offset of the other. In a graph, calculating distance is much more complicated; there may be multiple paths that connect the two positions and different paths may be relevant for different problems.

Distance is a basic function that is necessary for many functions on genome graphs; in particular, calculating distance is essential for efficient mapping algorithms. In a seed-and-extend paradigm, short seed matches between the query sequence and reference are used to identify small regions for expensive alignment algorithms to align to [75, 54, 69, 37, 88, 71]. Often these regions are identified by clusters of matches. Clustering requires repeated distance calculations between seeds and can be very slow in graphs as large as whole genome graphs. The prohibitive run time of clustering algorithms can make them impractical for mapping and

some mapping algorithms omit this step entirely [71].

We have developed an algorithm to calculate the exact minimum distance between any two positions in a sequence graph and designed an index to support it. We also developed a clustering algorithm that clusters seeds based on the minimum distance between them. Our algorithms are implemented as part of vg, a variation graph toolkit [37].

### 3.3 Background

#### 3.3.1 Sequence Graph Structure

A sequence graph is a bidirected graph in which each node is labeled by a sequence of nucleotides. A node  $X$  has two sides,  $\{x, \bar{x}\}$ . For convenience, we will consider  $x$  to be the “left” side and  $\bar{x}$  to be the “right”. This induces a directionality on  $X$ , so that we may consider a left-to-right (or  $x$  to  $\bar{x}$ ) traversal of  $X$  to be forward, and a right-to-left traversal backward. However, we note that the designation of “left” and “right” is arbitrary. They can be swapped without changing the underlying graph formalism. Conceptually, a forward traversal corresponds to the forward strand, and a backward traversal corresponds to the reverse complement strand.

Paths in a bidirected graph must obey restrictions on both nodes and edges. Edges connect two node sides rather than nodes. A path consists of an alternating series of oriented nodes and edges. The path must enter and exit each (non-terminal) node through opposite node sides. In addition, there must exist an edge connecting consecutive nodes in the path, between the node side that is exited and the node side that is entered.

In Figure 3.1, the graph has an edge between  $\bar{a}$  and  $b$ . A path including this edge

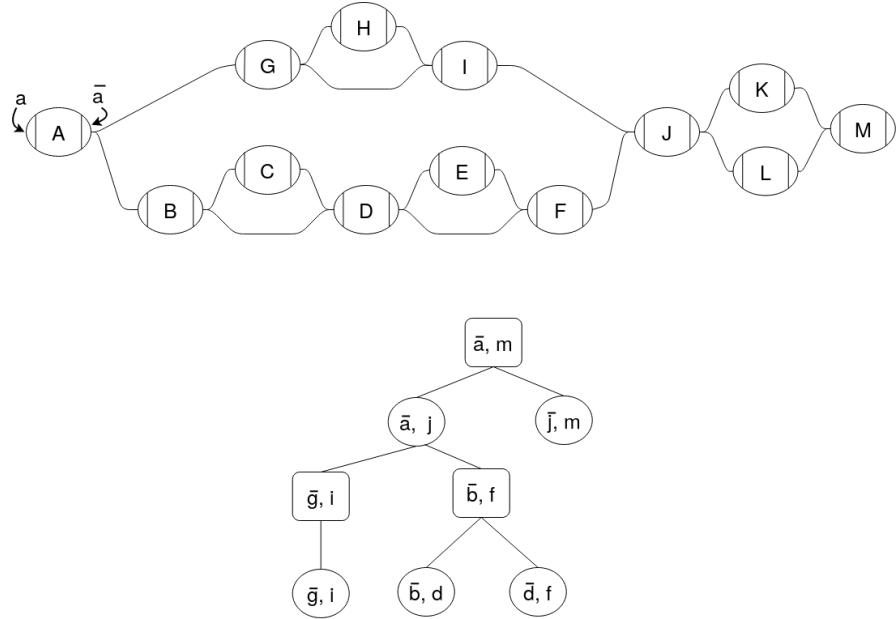


Figure 3.1: Example sequence graph (top) and its snarl tree (bottom). Chains in the sequence graph are represented as rectangular nodes in the snarl tree and snarls are represented as elliptical nodes.

would go from  $A$  to  $B$  traversing both forward, or from  $B$  to  $A$  traversing both backward.

Some applications use a specific articulation of a sequence graph called a variation graph. A variation graph contains a set of embedded paths through the graph. These paths typically correspond to the primary and alternate scaffolds of a reference genome.

### 3.3.2 Snarl Decomposition

In previous work, we proposed a decomposition for sequence graphs that describes their common topological features [63]. A simple variant, such as an indel or SNP, will typically be represented as one or two nodes (corresponding to the different alleles), flanked by two more nodes (corresponding to adjacent conserved sequences. In Figure 3.1, nodes  $A$ ,  $J$ , and  $M$  all represent conserved sequences. Nodes  $K$  and  $L$  represent two alternative sequences that occur

between  $J$  and  $M$ . The subgraph between the two flanking nodes, in this case the subgraph containing nodes  $J, K, L$ , and  $M$ , is called a snarl. Snarls can be seen as a generalization of the variant ‘bubbles’ used in many genome assembly algorithms [63].

A snarl is defined by a pair of node sides,  $(x, y)$  that delimit a subgraph between them. The nodes  $X$  and  $Y$  are called the boundary nodes of the snarl. Two node sides define a snarl if they are (i) separable: splitting the boundary nodes into their two node sides disconnects the snarl from the rest of the graph, and (ii) minimal: there is no node  $A$  in the snarl such that  $(x, a)$  or  $(\bar{a}, y)$  are separable. In Figure 3.1,  $\bar{g}$  and  $i$  define a snarl  $(\bar{g}, i)$ . We will sometimes abuse the terminology and use “snarl” to refer to both the pair of nodes and the subgraph that they separate from the rest of the graph. Thus, we can say that the snarl  $(\bar{g}, i)$  contains node  $H$  and boundary nodes  $G$  and  $I$ .

In sequence graphs, snarls often occur contiguously with a shared boundary node between them; a sequence of contiguous snarls is called a chain. In Figure 3.1, the snarls  $(\bar{b}, d)$  and  $(\bar{d}, f)$  comprise a chain between  $\bar{b}$  and  $f$ , which we refer to as  $[\bar{b}, f]$ . A trivial chain is one that contains only one snarl; in Figure 3.1, snarl  $(\bar{g}, i)$  is part of a trivial chain, chain  $[\bar{g}, i]$ .

Snarls and chains can be nested within other snarls. This nesting behavior often occurs when the same genomic element is affected by both point and structural variants, in which case the point variant’s snarl nests inside the structural variant’s snarl. A snarl  $(x, y)$  contains another snarl  $(a, b)$  if all nodes in  $(a, b)$  are contained in the subgraph of  $(x, y)$ . In Figure 3.1, the snarl  $(\bar{a}, j)$  contains snarls  $(\bar{g}, i)$ ,  $(\bar{b}, d)$ , and  $(\bar{d}, f)$ . A snarl contains a chain if each of the chain’s snarls are in the subgraph of the containing snarl.

The nesting relationships of snarls and chains in a sequence graph is described by its

snarl tree (Figure 3.1). Each snarl or chain is represented in the snarl tree as a node. Since every snarl belongs to a (possibly trivial) chain, snarl trees have alternating levels of snarls and chains with a chain at the root of the tree. We also refer to the root as the top-level chain. A snarl is the child of a chain if it is a component of the chain. A chain  $[a, b]$  is a child of  $(x, y)$  if  $(x, y)$  contains  $[a, b]$  and there are no snarls contained in  $(x, y)$  that also contain  $[a, b]$ .

All nodes in a sequence graph will be contained by the decomposition of its snarls and chains, described by the snarl tree. In general, the snarl tree can be arbitrarily deep and have very short chains. However the snarl tree of a typical sequence graph will be shallow and have a long chain as the root. The majority of snarls will be contained in this top-level chain. Small variants can nest within larger structural variants, contributing to the depth of the snarl tree. However, in most parts of the genome, the rate of polymorphism is low enough that two variants are unlikely to overlap each other. As a result, the depth of these nested variants is usually very small, typically less than 5 in our observations.

Nodes, snarls, and chains are all two-ended structures that are connected to the rest of the graph by two node sides. It is sometimes convenient to refer to a topological feature only by this shared property, and to be opaque about which topological feature it actually is. In these cases, we will refer to the node, snarl, or chain generically as a “structure”. As with nodes of the sequence graph itself, structures are assigned an arbitrary orientation but we will assume that they are oriented left to right and refer to the left and right sides of structures as *struct* and *struct̄* respectively. Because of their shared two-ended property, structures can all be treated as single nodes in their parents. The netgraph of a snarl is a view of the snarl where each of its child chains is replaced by a node.

### 3.3.3 Prior Research

#### 3.3.3.1 Distance in graphs

Calculating distance in a graph is an extremely well studied topic. Many graph distance algorithms improve upon classical algorithms, such as Dijkstra's algorithm [26] and A\* [40], by storing precomputed data in indexes. These methods index the identities of important edges [58, 51] or distances between selected nodes [24, 68, 27, 7] then use the indexed information to speed up distance calculations. Index-based algorithms must make a tradeoff between the size of the index and the speed of the distance query.

#### 3.3.3.2 Distance in sequence graphs

Some sequence graph mapping algorithms use clustering steps based on different estimations of distance [88, 37]. In vg, distance is approximated from the embedded paths. This path-based method estimates the distance between two positions based on a nearby shared path. The algorithm performs a bidirectional Dijkstra search from both positions until it finds at least one path in common from both positions. This path is then used to estimate the distance between them.

Some research has been done on finding solutions for more specific distance queries in sequence graphs. PairG [45] is a method for determining the validity of independent mappings of reads in a pair by deciding whether there is a path between the mappings whose distance is within a given range. This algorithm uses an index to determine if there is a valid path between two vertices in a single O(1) lookup. Although this is an efficient solution for this particular

problem, it cannot be used to query the exact distance between two nodes. Rather, it returns a boolean value indicating whether two nodes are reachable within a range of distances, which is defined at index construction time.

### 3.4 Minimum Distance

Our minimum distance algorithm finds the minimum oriented traversal distance between two positions on a sequence graph. A position consists of a node, offset in the sequence, and orientation. The oriented distance must originate from a path that starts traversing the first position in its given orientation and ends at the second position in its given orientation.

Our algorithm uses the snarl decomposition of sequence graphs to guide the calculation. Because structures are connected to the rest of the graph by their boundary nodes, any path from a node inside a structure to any node not in that structure must pass through the structure's boundary nodes. Similarly, any path between boundary nodes of snarls in a chain must pass through the boundary nodes of every snarl that occurs between them in the chain. Because of this property, we can break up the minimum distance calculation into minimum distances from node and chain boundaries to the boundaries of their parent snarl, from snarl boundaries to their parent chain boundaries, and the distance between sibling structures in their parent structure (Figure 3.2). We refer to this property of minimum distance calculation in structures as the split distance property.

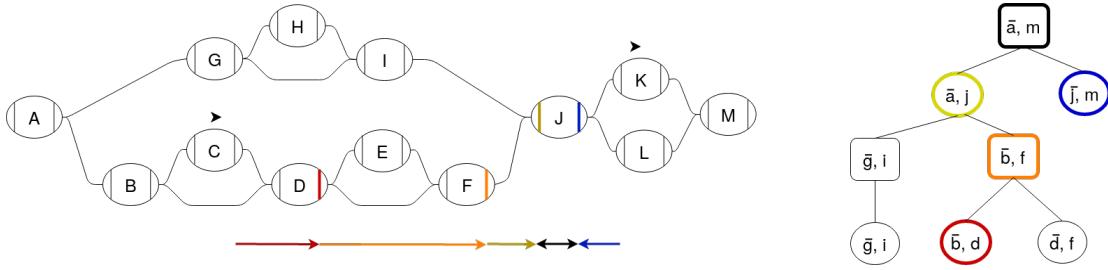


Figure 3.2: The minimum distance calculation from a position on  $C$  to a position on  $K$  can be broken up into the distances from each position to the ends of each of its ancestor structures in the snarl tree. Each colored arrow in the graph represents a distance query from a structure to a boundary node of its parent. The snarl tree node that each query occurs in is outlined with the same color. At the common ancestor of the positions, chain  $[\bar{a}, m]$ , the distance is calculated between two of the chain's children,  $(\bar{a}, j)$  and  $(\bar{i}, m)$ .

### 3.4.1 Minimum Distance Index

We designed our minimum distance index to support distance queries between child structures in snarls and between boundary nodes of snarls in chains in constant time. The overall minimum distance index consists of a snarl index for each snarl and a chain index for each chain in the graph.

#### 3.4.1.1 Snarl Index

For each snarl, the index stores the minimum distances between every pair of node sides of child structures contained in the snarl, including the boundary nodes. A distance query within a snarl is a simple constant time lookup of the distance.

#### 3.4.1.2 Chain Index

For each chain, the index stores three arrays, each with one entry for each boundary node of the snarls in the chain. The first, a prefix sum array, contains the minimum distance from

the start of the chain to the left side of each of the boundary nodes of the snarls that comprise the chain. This array can be used to find the distance between two of these snarls' boundary nodes along the chain. Distances from a left-to-right traversal of the chain can be computed directly from the prefix sum array, whereas distances from a right-to-left traversal also require the length of the boundary nodes. Since paths can reverse direction in the chain (Figure 3.3a), the index also stores each boundary node's "loop distance". The loop distance is the minimum distance to leave a boundary node, change direction in the chain, and return to the same node side traversing in the opposite direction. These loop distances are stored in final two arrays, one for each direction. In Figure 3.3a, the forward loop distance for node  $C$  is two times the length of  $E$ : the distance to leave  $\bar{c}$  traversing forward and return to  $\bar{c}$  traversing backward by taking the bold looping edge on  $\bar{e}$ . These three arrays are sufficient to find the minimum distance between any two node sides in the chain in constant time (Figure 3.3).

Chains that are not top-level chains cannot form a closed cycle so any path that traverses a chain's boundary node going out of the chain must leave the chain. Therefore any connectivity between the boundaries of the chain will be captured by the snarl index of the chain's parent. The top-level chain may form a closed cycle where the start and end boundary nodes are the same node (Figure 3.4). In this case, the shortest path may remain within the chain, but it may also leave the chain and re-enter it from the other side. In Figure 3.4, the minimum distance from  $\bar{a}$  to  $d$  could be  $d(\bar{a}, \bar{d}) + d(d, d)$  or  $d(\bar{a}, \bar{a}) + d(a, d)$ .

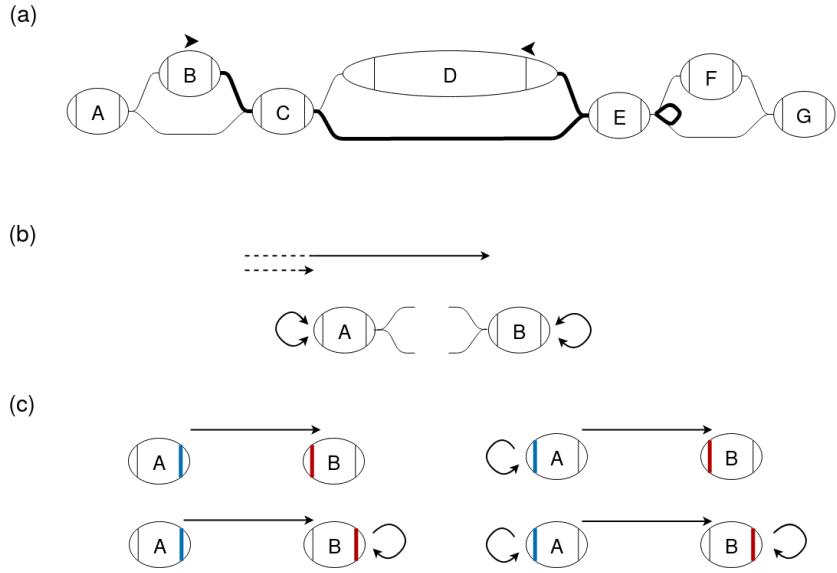


Figure 3.3: (a) The shortest path between two nodes in a chain can sometimes reverse direction in the chain. The edges on the shortest path between the positions on  $B$  and  $D$  are bolded. (b)  $A$  and  $B$  are boundary nodes of snarls in a chain. Distances stored in the chain index are shown in black. For each boundary node in the chain, the chain index stores the minimum distance from the start of the chain to the left side of that node as well as the loop distances for a forward and backward traversal. These loop distances are the minimum distance to leave a node, reverse direction in the chain, and return to the same node side. (c) There are four possible minimum-distance paths between two nodes, connecting either node side of the two nodes. The lengths of these paths can be found using the distances stored in the chain index and the lengths of the nodes.

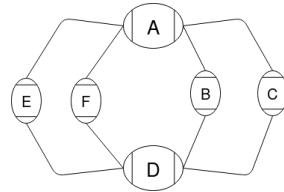


Figure 3.4: A cyclic chain containing two snarls,  $(\bar{a}, \bar{d})$  and  $(d, a)$

### 3.4.1.3 Index Construction

The minimum distance index is constructed in a post-order traversal of the snarl tree. For each snarl, the construction algorithm does a Dijkstra traversal starting from each child structure, using the child's index to find the distance to traverse child snarls or chains. For each chain, the construction algorithm traverses through each snarl in the chain and uses the snarl's index to find each of the relevant distances for the chain index.

### 3.4.1.4 Index Size

Naively, a minimum distance index could store the minimum distance between every node in the graph. A distance calculation would be a constant time lookup but the index size would be quadratic in the number of nodes in the graph. For each snarl in the graph, our index stores the distance between every pair of structures in the net graph. For each chain, it stores three arrays, each the length of the chain. In a graph with a set of snarls  $S$  and chains  $C$ , our index will take  $O(\sum_S n_s^2 + \sum_C n_c)$  space where  $n_s$  is the number of structures in the netgraph of snarl  $s$  and  $n_c$  is the number of snarls in chain  $c$ .

## 3.4.2 Minimum Distance Algorithm

The first step of our minimum distance algorithm (Algorithm 2) is to find the least common ancestor structure in the snarl tree that contains both positions. We do this by traversing up the snarl tree from each position and finding the first common structure. This traversal is  $O(d)$  where  $d$  is the depth of the snarl tree.

Next, the algorithm finds the distance from each position to the ends of the child of the

least common ancestor (Algorithm 1). Starting at a position on a node, we find the distances to the ends of the node. If both positions are oriented forward, then we find the distance to the right side of the first node and the left side of the second, and we record the distances to the opposite sides as infinite. In the case where a position is oriented backward, we find the distance to the opposite side. The algorithm then traverses up the snarl tree to the least common ancestor and at each structure, finds the minimum distances to the ends of the structure. Because of the split distance property, this distance can be found by adding the distances to the ends of the child, found in the previous step in the traversal, to the distances from the child to the boundary nodes of the structure, found using the minimum distance index (Figure 3.5). Since this requires only four constant-time queries to the minimum distance index, each step in the traversal is constant time and the overall traversal is  $O(d)$ .

At this point in the algorithm, we know the minimum distance from each position to its ancestor structure that is a child of the common ancestor. By composing these distances with the distances between the two structures, the algorithm finds possible distances between the two positions in the common ancestor structure. The algorithm continues to traverse the snarl tree up to the root and finds a minimum distance between the positions at each structure, checking for paths that leave the lowest common ancestor. This traversal is also  $O(d)$ . The minimum distance algorithm is done in three  $O(d)$  traversals of the snarl tree, so the algorithm is  $O(d)$ . In variation graphs for moderately large genomes without extreme levels of polymorphism, snarl trees are very shallow. In these graphs, the algorithm is expected to be  $O(1)$ . However, for variation graphs of small, highly polymorphic genomes, the run time may grow with increasing amounts of population variation. Complex sequence graphs derived from assembly graphs also

may demonstrate slower run time behavior.

Table 3.1: Primitive functions for the minimum distance algorithm

Function	Description	Complexity
<code>distToEndsOfParent(     <i>struct</i>,     <i>dist_left</i>,     <i>dist_right</i>)</code>	Given the distances from a position in a structure <i>struct</i> to the ends of <i>struct</i> , find the distance to the ends of the parent (Figure 3.5)	$O(1)$ using the distance index
<code>distWithinStructure(     <i>struct</i>,     <i>child_1</i>,     <i>child_2</i>,     <i>dist1_l</i>,     <i>dist1_r</i>,     <i>dist2_l</i>,     <i>dist2_r</i>)</code>	Given two children of a structure and distances from positions to the boundaries of the children, find the minimum distance between the positions in <i>struct</i>	$O(1)$ using the distance index

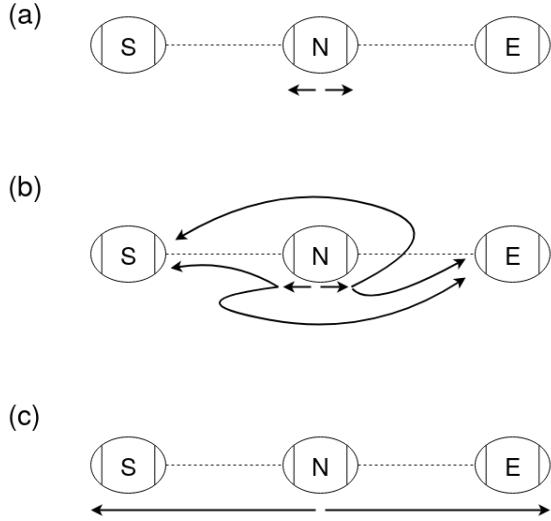


Figure 3.5: The `distToEndsOfParent` calculation described in Table 3.1. (a)  $S$  and  $E$  are the boundary nodes of a structure that contains a child structure  $N$ . The minimum distances from some object in  $N$  to the ends of  $N$  shown as black arrows. (b) The minimum distances from each end of  $N$  to  $s$  and  $e$  are found using the minimum distance index. (c) By adding the appropriate distances and taking the minimums, we can get the minimum distances to  $s$  and  $e$ .

---

**Algorithm 1:** `distToAncestor(position, ancestor)`: Given a position and ancestor structure, return the minimum distance from the position to both sides of a child of the ancestor and the child

---

```

begin
  struct  $\leftarrow$  parentOf(position)
  dist_l, dist_r  $\leftarrow$  distances from position to ends of node, one is  $\infty$ 
  while parentOf(struct) is not ancestor do
    /* Find the minimum distance from position to the
       boundaries of each ancestor */
    dist_l, dist_r  $\leftarrow$  distToEndsOfParent(struct, dist_l, dist_r)
    struct  $\leftarrow$  parentOf(struct)
  return dist_l, dist_r, struct

```

---

---

**Algorithm 2:**  $\text{minDistance}(\text{position\_1}, \text{position\_2})$ : Return the minimum distance from  $\text{position\_1}$  to  $\text{position\_2}$ ,  $\infty$  if no path between them exists

---

```

begin
    /* Get distances from each position to the ends of a child
       of the least common ancestor */  

    ancestor  $\leftarrow$  leastCommonAncestor(position_1, position_2)
    dist1_l, dist1_r, struct_1  $\leftarrow$  distToAncestor(position_1, ancestor)
    dist2_l, dist2_r, struct_2  $\leftarrow$  distToAncestor(position_2, ancestor)
    min_dist  $\leftarrow$   $\infty$ 
    while ancestor is not root of snarl tree do
        /* Given the distance from each position to both sides of
           a child of ancestor, find the minimum distance between
           the two positions in ancestor */  

        min_dist  $\leftarrow$  min(min_dist, distWithinStructure(ancestor, struct_1,
            struct_2, dist1_l, dist1_r, dist2_l, dist2_r))
        dist1_l, dist1_r  $\leftarrow$  distToEndsOfParent(struct_1, dist1_l, dist1_r)
        dist2_l, dist2_r  $\leftarrow$  distToEndsOfParent(struct_2, dist2_l, dist2_r)
        struct_1  $\leftarrow$  ancestor, struct_2  $\leftarrow$  ancestor
        ancestor  $\leftarrow$  parentOf(ancestor)
    return min_dist

```

---

## 3.5 Clustering

Seed-and-extend algorithms sometimes cluster seed alignments by their location in the graph to find which might belong to the same mapping. Using our minimum distance index, we developed an algorithm to cluster positions based on the minimum distance between them in the graph.

### 3.5.1 Problem

We will cluster seeds by partitioning them based on the minimum distance between their positions in a sequence graph. To define a cluster, we consider a graph where each seed is a node and two seeds are connected if the minimum distance between their positions is less

than a given distance limit. In this graph, each connected component is a cluster.

### 3.5.2 Algorithm

Our clustering algorithm starts with each position in a separate cluster then progressively agglomerates the clusters (Figure 3.6). The algorithm proceeds in a post-order traversal of the snarl tree and, at each structure, produces clusters of all positions contained in that structure (Algorithm 5). After iterating over a structure, clusters are also annotated with two “boundary distances”: the shortest distance from any of its positions to the boundary nodes of the structure. At every iteration, each cluster can be unambiguously identified with a structure and so the boundary distances are always measured to the structure the cluster is on.

The method of agglomerating clusters and computing boundary distances vary according to the type of structure. For nodes, the algorithm creates a sorted array of the positions contained in it and splits the array into separate clusters when the distance between successive positions is large enough. For each new cluster, the boundary distances are computed from the positions’ offsets.

For structures that are snarls or chains, clusters are created from the clusters on their children (Algorithm 3, Algorithm 4). Clusters associated with child structures are compared and if the distance between any pair of their positions is smaller than the distance limit, they are combined. Within a structure, distances to clusters that are associated with child structures can be calculated using the split distance property as in the minimum distance algorithm. According to this property, the minimum distance can be split into the cluster’s boundary distance and the distance to one of the boundary nodes, which is found using the index. For snarls, all pairs of

clusters are compared to each other. For chains, clusters are combined in the order they occur in the chain, so each cluster is compared to agglomerated clusters that preceded it in the chain. Finally, for each of the resulting clusters, we compute the boundary distances for the current structure, once again using the boundary distances of the children and the index.

In the worst case, every position would belong to a separate cluster and at every level of the snarl tree, every cluster would be compared to every other cluster. This would be  $O(dn^2)$  where  $d$  is the depth of the snarl tree and  $n$  is the number of seeds, so in the worst case our clustering algorithm is no better than the naive algorithm of comparing every pair of positions with our minimum distance algorithm. In practice, however, seeds that came from the same alignment would be near each other on the graph and form clusters together, significantly reducing the number of distance comparisons that would be made.

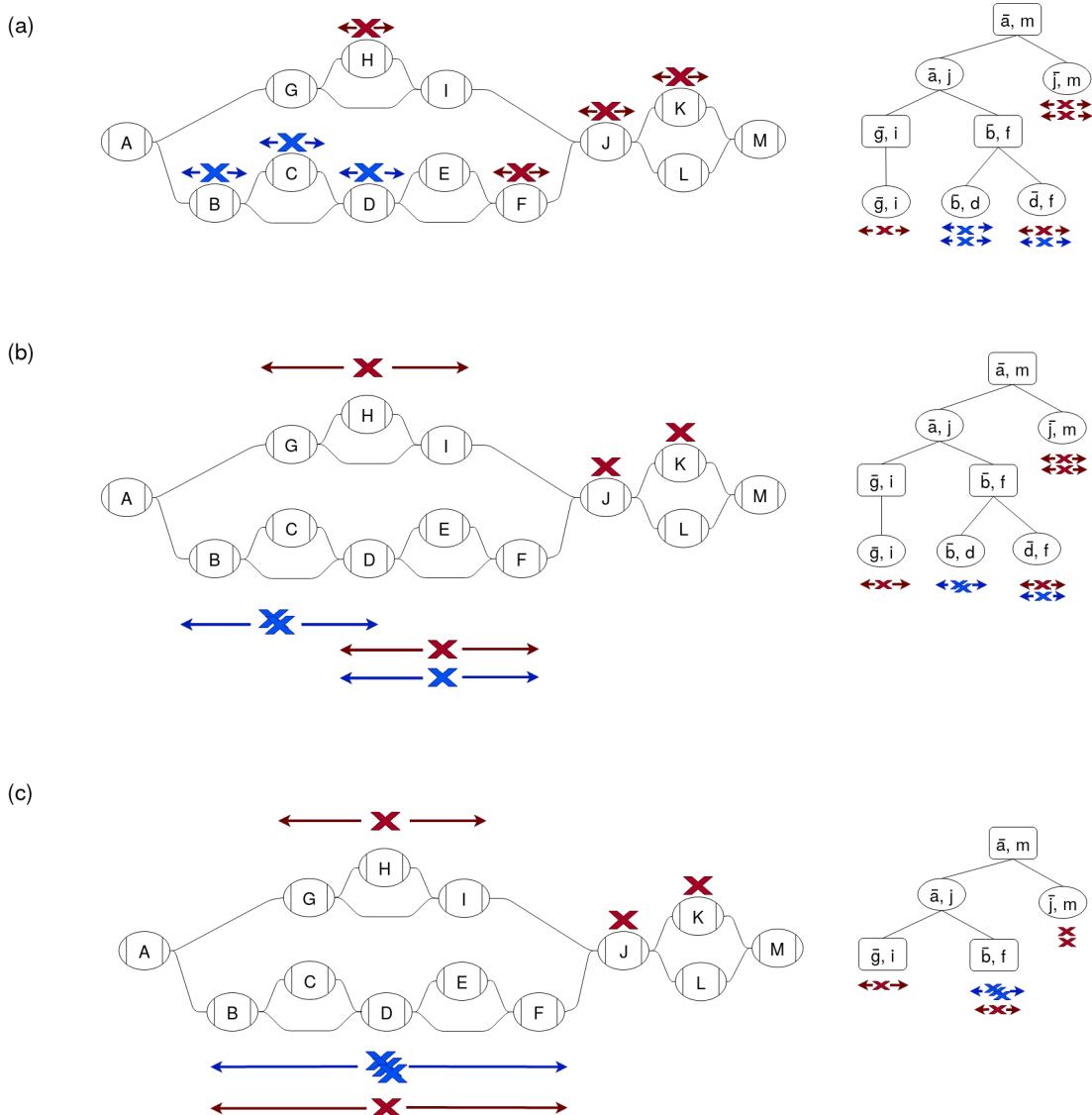


Figure 3.6: Clustering of positions (Xs) is done by traversing up the snarl tree and progressively agglomerating clusters. Positions are colored by the final clusters. (a) Each position starts out in a separate cluster on a node. Each cluster is annotated with its boundary distances: the minimum distances from any of its positions to the ends of the structure it is on. (b) For each snarl on the lowest level of the snarl tree, the clusters on the snarl’s children are agglomerated into new clusters on the snarl. The boundary distances are extended to the ends of the snarl. (c) For each chain on the next level of the snarl tree, the clusters on the chain’s snarls are agglomerated and the boundary distances are updated to reach the ends of the chain. This process is repeated on each level of the snarl tree up to the root.

---

**Algorithm 3:** clusterSnarl(*snarl*, *child\_to\_clusters*, *distance\_limit*): Given a snarl and map from children of the snarl to their clusters, get clusters of the snarl

---

```
begin
    snarl_clusters ← // Array of all clusters in child_to_clusters
    for struct,cluster in child_to_clusters do
        /* Record the minimum distances from each cluster to the
           boundaries of snarl */
        cluster.dist_left_parent,cluster.dist_right_parent ←
            distToEndsOfParent(struct,cluster.dist_left,cluster.dist_right)
    for struct_1,cluster_1 in child_to_clusters do
        for struct_2,cluster_2 in child_to_clusters do
            /* Compare each pair of clusters and if they are close
               enough, combine them */
            cluster_dist ← distWithinStructure(snarl, struct_1, struct_2,
                cluster_1.dist_left, cluster_1.dist_right, cluster_2.dist_left,
                cluster_2.dist_right)
            if cluster_dist ≤ distance_limit then
                Agglomerate cluster_1 and cluster_2, take the minimum
                dist_left_parent and dist_right_parent
    for cluster in snarl_clusters do
        /* Update boundary distances to reach the ends of snarl */
        cluster.dist_left,cluster.dist_right ←
            cluster.dist_left_parent,cluster.dist_right_parent
return snarl_clusters
```

---

---

**Algorithm 4:** clusterChain(*chain*, *child\_to\_clusters*, *distance\_limit*): Given a chain and a map from each snarl in the chain to its clusters, get clusters of the chain

---

```
begin
    chain_clusters ← [] // Empty array of clusters of chain
    for snarl, snarl_cluster in child_to_clusters do
        /* Record the minimum distances from snarl_cluster to the
           boundaries of chain */
        snarl_cluster.dist_left_parent, snarl_cluster.dist_right_parent ←
            distToEndsOfParent(snarl, cluster.dist_left, dist_right) for
            chain_cluster in chain_clusters do
                /* Compare the snarl clusters with each previously
                   found chain cluster */
                if chain_cluster.distance_right + snarl_cluster.distance_left ≤
                    distance_limit then
                    Agglomerate snarl_cluster and chain_cluster, take the minimum
                    dist_left_parent and dist_right_parent
            for cluster in chain_clusters do
                /* Update the right distance of each cluster to reach
                   the end of snarl */
                cluster.dist_right ← cluster.dist_right + snarl.length
            Add any uncombined snarl clusters to chain_clusters
    return chain_clusters
```

---

---

**Algorithm 5:** *cluster(snarl\_tree, positions, distance\_limit)*: Cluster positions based on the distance limit

---

```
begin
    struct_to_clusters // Map each structure to its clusters
    for struct in snarl_tree do
        /* Traverse structures in post-order */ *
        if struct is a node then
            struct_to_clusters[struct] ← clusters of positions on struct
        else if struct is a snarl then
            child_clusters ← // Get map from each child of struct to
                            its clusters
            struct_to_clusters[struct] ← clusterSnarl(struct, child_clusters,
                                            distance_limit)
        else
            child_clusters ← // Get map from each child of struct to
                            its clusters
            struct_to_clusters[struct] ←
                clusterChain(struct, child_clusters, distance_limit)
    return struct_to_clusters[snarl_tree.root]
```

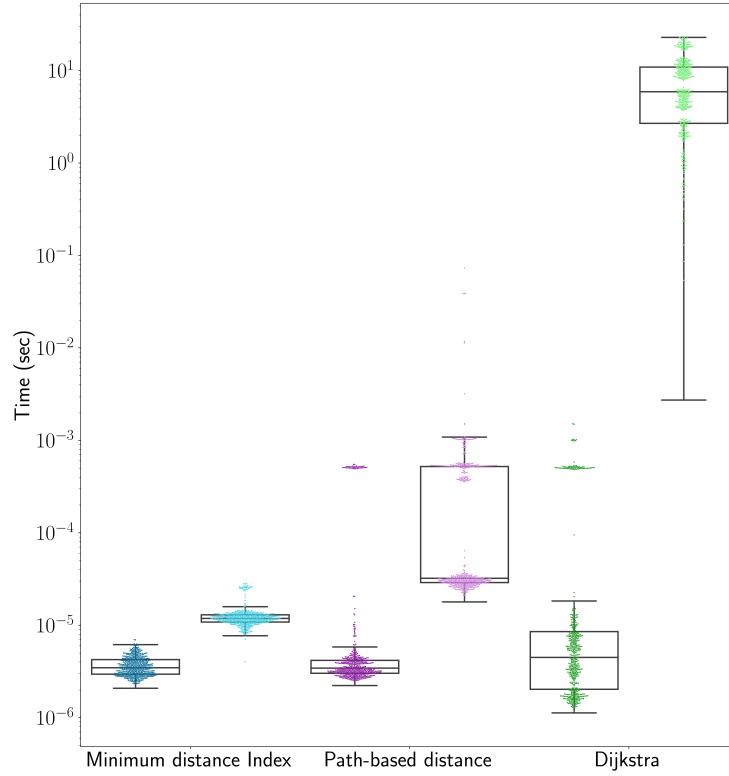
---

### 3.6 Methods and Results

Our algorithms are implemented as part of the `vg` toolkit. We conducted experiments on two different graphs: a human genome variation graph and a graph with simulated structural variants. The human genome variation graph was constructed from GRCh37 and the variants from the 1000 Genomes Project. The structural variant graph was simulated with 10bp-1kbp insertions and deletions every 500bp.

The human genome variation graph had 306,009,792 nodes, 396,177,818 edges, and 3,180,963,531 bps of sequence. The snarl tree for this graph had a maximum depth of three snarls with 139,418,023 snarls and 11,941 chains. The minimum distance index for the graph was 12.2 GB on disk and 17.7 GB in memory.

To assess the run time of our minimum distance algorithm, we calculated distances between positions on the whole genome graph and compared the run time of our algorithm to `vg`'s path-based algorithm and Dijkstra's algorithm (Figure 3.7). We chose random pairs of positions in two ways. The first method sampled positions uniformly at random throughout the graph. The second method first followed a random walk of 148 bp through the graph and then sampled two positions uniformly at random from this random walk. This approach was intended to approximate the case of seeds from a next-generation sequencing read. On average, our minimum distance algorithm is the fastest of the three algorithms for both sets of positions. In addition, all three algorithms' performance degraded when the positions could be sampled arbitrarily far apart in the graph, but our minimum distance algorithm's performance degraded the least.

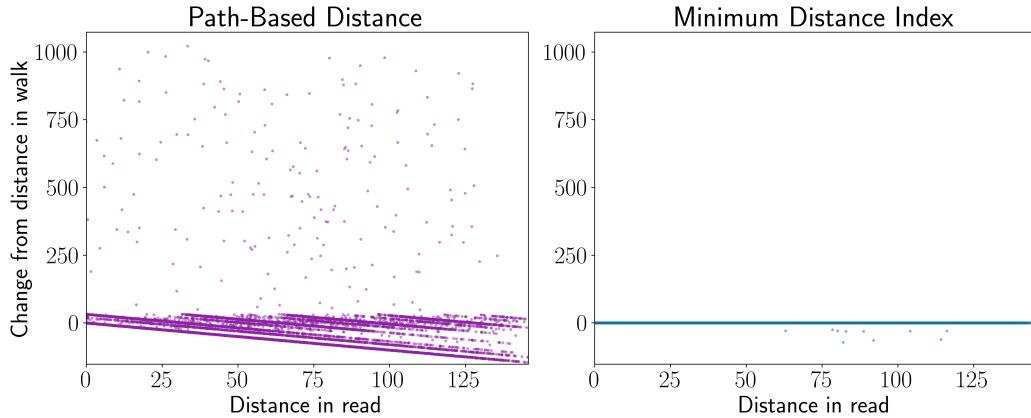


**Figure 3.7: Run times for distance algorithms.** Random pairs of positions were chosen from either within a read-length random walk (dark colors) or randomly from the graph (light colors).

Our new minimum distance algorithm shows a distinct gain in performance over the other methods, however the algorithm must trade off speed with the memory consumed by the index. A hybrid approach could be imagined where the index is used to compute the distance up structures in the common ancestor, then Dijkstra's algorithm could be used to connect the structures. The runtime of such a hybrid algorithm is in the worst case the same as Dijkstra's algorithm. Using this approach, the distance index would only need to store the distance from each node in a snarl to the boundary nodes of the snarl, rather than the distance between every pair of nodes, reducing the memory requirement of the index.

In the context of read mapping, we are often only interested in the exact distance when the minimum distance is small, but when the minimum distance is large enough the exact distance is not necessary. In this scenario, the algorithm could be accelerated by stopping early when it is apparent that the minimum distance will be too large.

We used the structural variant graph to assess whether the minimum distance is a useful measure of distance for read mapping. We compared our minimum distance algorithm to the path-based approximation, which estimates distances based on linear paths corresponding to scaffolds of a reference genome. To do so, we again used read-length random walks to select pairs of positions. Further, we filtered random walks down to those that overlapped a structural variant breakpoint. We then calculated the distances between pairs of positions using our minimum distance algorithm and the path-based approximation and compared these distances to the actual distances in the random walk, which we take as an approximation of the true distance on a sequencing read. Overall, the minimum distance was a much better estimate of distance along the random walk than the path-based distance approximation (Figure 3.8).



**Figure 3.8: Distance calculations on a graph with simulated structural variants.** Read-length random walks were simulated near the junctions of structural variants. The distance between two random positions along each walk was calculated using the path-based method and our minimum distance algorithm and compared to the actual distance in the walk.

For our clustering algorithm, we wanted to estimate the run time of the algorithm in the context of read mapping. We simulated 148bp reads from AshkenazimTrio HG002\_NA24385\_son from the Genome in a Bottle Consortium [92]. For each read, we sampled 15-mer matches from the read and found their positions in the human genome variation graph using a  $k$ -mer lookup table. We then apply the clustering algorithm to the positions of these  $k$ -mers. The regression line of the log-log plot of run times suggests the run time of our algorithm is linear in the number of positions in practice, despite the quadratic worst-case bound (Figure 3.9).

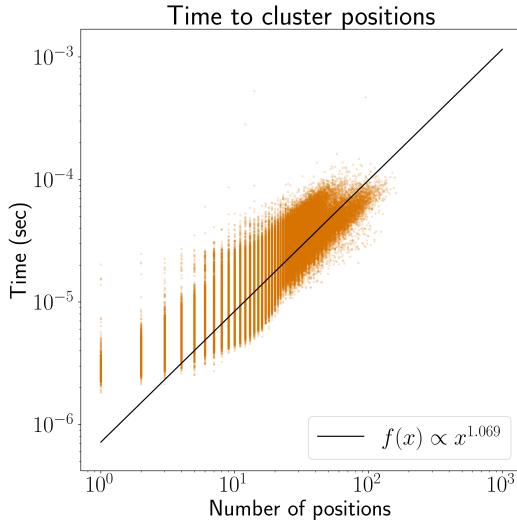


Figure 3.9: **Run time growth of our clustering algorithm.** The regression line suggests that the run time of our algorithm is approximately linear in the number of positions in practice.

### 3.7 Conclusion

Pangenomes have the potential to eliminate reference bias and grow the inclusiveness of reference structures used in genomics, but substantial algorithmic challenges remain in adapting existing paradigms to use them. We have developed a simple and elegant minimum distance algorithm with run time that is linear in the depth of the snarl tree. In practice, the algorithm exploits the observation that real-world genome graphs have an excess of small, local variations and relatively fewer variations that connect disparate parts of the graph. The result is that real genome graphs have a shallow snarl tree, making the calculations fast and effectively constant time in practice; indeed, we observe the algorithm is substantially faster than other distance algorithms on queries of arbitrary distance. The minimum distance we return is an exact distance, unlike the previous heuristic implementation of distance in `vg`, resulting in much more

reasonable estimates of distance around the breakpoints of structural variants. Our minimum distance algorithm will also work with any sequence graph, whereas the preexisting `vg` distance algorithm required pre-specified paths. Here we developed a clustering algorithm for clustering positions on the graph based on the minimum distances between them. Clustering is a major component of many mapping algorithms and calculating distance is a bottleneck of clustering in genome graphs. Our new clustering algorithm runs in linear time relative to the number of seeds, whereas many existing algorithms, including the current `vg` mapper’s path-based algorithm, are (at least) quadratic due to pairwise distance calculations between seeds. We believe this is an important step in generalizing efficient mapping algorithms to work with genome graphs; we are now developing fast mapping algorithms that use this clustering algorithm.

### 3.8 Funding

This work was supported, in part, by the National Institutes of Health (award numbers: 5U54HG007990, 5T32HG008345-04, 1U01HL137183, R01HG010053, U01HL137183, 2U41HG007234).

# **Chapter 4**

## **Short read mapping**

### **4.1 Preface**

This chapter contains the full main text of the paper "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes"[[80](#)], published in Science in 2021. It also includes portions of the supplement that describe my contributions or that are referenced in the paper. The text contains minor edits to conform with the format of this thesis.

I was a co-first author along with Jouni Sirén, Jean Monlong, Adam M. Novak, and Jordan M. Eizenga. Jouni Sirén, Adam M. Novak, and I did the majority of the software development of Giraffe. I integrated the distance index and clustering algorithm from the previous chapter into the Giraffe implementation, adapted the single end algorithm to paired-end mapping, contributed to the paired-end rescue implementation, and contributed to the overall software development and optimization of the Giraffe tool. Jouni Sirén implemented the GBWT data structure and related algorithms including the minimizer index, seeding, haplotype sam-

pling, and gapless extension. Adam M. Novak integrated the components of Giraffe into the full working tool, built the graphs for evaluations, and contributed to the optimization of Giraffe. Jordan M. Eizenga wrote the alignment algorithms and mapping quality code that Giraffe uses. Jean Monlong performed all the structural variant analyses. I performed the simulated read mapping experiments, runtime and memory analyses, and reference bias analyses. Charlie Markello performed the short variant calling analyses. Jonas A. Sibbesen built the indexes for and ran HISAT2. Glenn Hickey contributed to the structural variant analyses and built the yeast graph. Jouni Sirén, Jean Monlong, Adam M. Novak, and I wrote the majority of the paper.

## 4.2 Structured abstract

### 4.2.1 Introduction

Modern genomics depends on inexpensive short-read sequencing. Sequenced reads up to a few hundred base pairs in length are computationally mapped to estimated source locations in a reference genome. These read mappings are used in myriad sequencing-based assays. For example, through a process called genotyping, mapped reads from a DNA sample can be used to infer the combination of alleles present at each site in the reference genome.

### 4.2.2 Rationale

A single reference genome cannot capture the diversity within even a single person (who gets a genome copy from each parent), let alone in the whole human population. Genomes differ not only by point variations, where one or a few bases are different, but also by structural

variations, where differences can be much larger than an individual read. When a person’s genome differs from the reference by a structural variation, the reference may contain no location to correctly map the corresponding reads. Although newer long-read sequencing allows structural variation to be more directly observed in sequencing reads, short-read sequencing is still less expensive and more widely available.

#### 4.2.3 Results

We present a short read–mapping tool, Giraffe. Giraffe maps to a pangenome reference that describes many genomes and the differences between them. Giraffe can accurately map reads to thousands of genomes embedded in a pangenome reference as quickly as existing tools map to a single reference genome. Simulations in which the true mapping for each read is known show that Giraffe is as accurate as the most accurate previously published tool. Giraffe achieves this speed and accuracy by using a variety of algorithmic techniques. In particular, and in contrast to previous tools, it focuses on mapping to the paths in the pangenome that are observed in individuals’ genomes: the reference haplotypes. This has two key benefits. First, it prioritizes alignments that are consistent with known sequences, avoiding combinations of alleles that are biologically unlikely. Second, it reduces the size of the problem by limiting the sequence space to which the reads could be aligned. This deals effectively with complex graph regions where most paths represent rare or nonexistent sequences.

Using Giraffe in place of a single reference genome reduces mapping bias, which is the tendency to incorrectly map reads that differ from the reference genome. Combining Giraffe with state-of-the-art genotyping algorithms demonstrates that Giraffe mappings produce

accurate genotyping results.

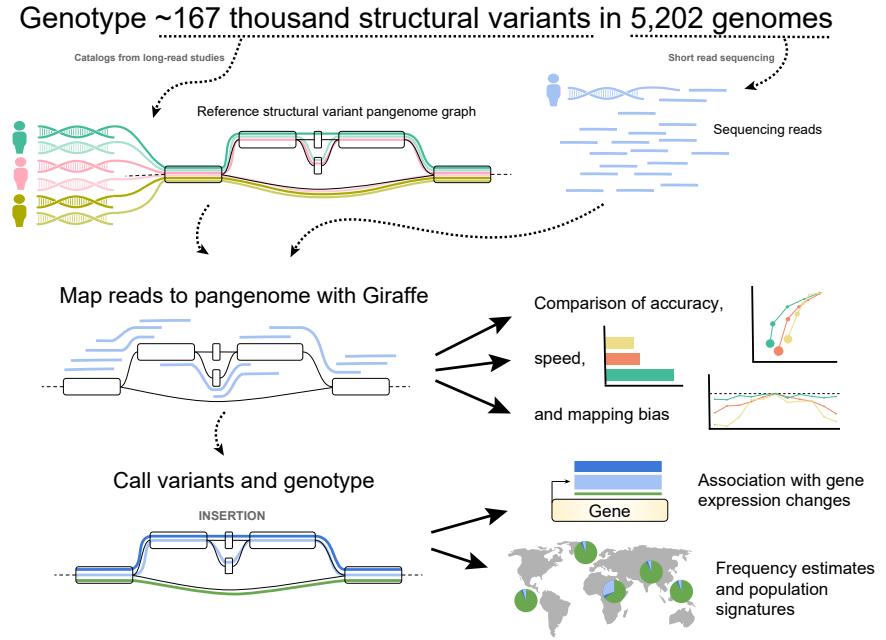
Using mappings from Giraffe, we genotyped 167,000 recently discovered structural variations in short-read samples for 5202 people at an average computational cost of \$1.50 per sample. We present estimates for the frequency of different versions of these structural variations in the human population as a whole and within individual subpopulations. We identify thousands of these structural variations as expression quantitative trait loci (eQTLs), which are associated with gene-expression levels.

#### 4.2.4 Conclusion

Giraffe demonstrates the practicality of a pangenomic approach to short-read mapping. This approach allows short-read data to genotype single-nucleotide variations, short insertions and deletions, and structural variations more accurately. For structural variations, this allowed the estimation of population frequencies across a diverse cohort of 5000 individuals. A single reference genome must choose one version of any variation to represent, leaving the other versions unrepresented. By making more broadly representative pangenome references practical, Giraffe attempts to make genomics more inclusive.

### 4.3 Abstract

We introduce Giraffe, a pangenome short-read mapper that can efficiently map to a collection of haplotypes threaded through a sequence graph. Giraffe maps sequencing reads to thousands of human genomes at a speed comparable to that of standard methods mapping to a



**Figure 4.1: Overview of the experiments** Variant calls from long read-based and large-scale sequencing studies were used to construct pangenome reference graphs (top). Giraffe (and competing mappers) mapped reads to the graph or to linear references, and mapping accuracy, allele coverage balance, and speed were evaluated (middle). Then, mapped reads were used for variant calling, and variant call accuracy was evaluated (bottom). Structural variant calls were analyzed alongside expression data to identify eQTLs and population frequency estimates.

single reference genome. The increased mapping accuracy enables downstream improvements in genome-wide genotyping pipelines for both small variants and larger structural variants. We used Giraffe to genotype 167,000 structural variants, discovered in long-read studies, in 5202 diverse human genomes that were sequenced using short reads. We conclude that pangenomics facilitates a more comprehensive characterization of variation and, as a result, has the potential to improve many genomic analyses.

## 4.4 Introduction

The field of genomics almost exclusively uses a single reference genome assembly as an archetype of a human genome. Reliance on comparing with the sequences within the reference assembly has created a pervasive bias toward the alleles it contains. This reference allele bias occurs because nonreference alleles are naturally harder to identify when mapping DNA sequencing data to the reference sequences. Reference allele bias is particularly acute for structural variations (SVs), which are complex alleles involving 50 or more nucleotides of divergent sequence. SVs affect millions of bases within each human genome. Because of reference allele bias, SVs are much more poorly characterized than single-nucleotide variants (SNVs) and short insertions and deletions (collectively termed indels)[[94](#), [57](#)]. Similarly, characterizing genetic variation in highly polymorphic and repetitive sequences has proven challenging [[30](#)].

Recent releases of the reference human genome assembly attempted to address these issues by adding additional sequences. These alternate sequences represent diversity in localized regions of the genome [[20](#)]. However, to date, these limited additions have not found widespread use. By contrast, pangenomes encode information about many complete genome assemblies and their homologies (the sequences that are shared between genomes by virtue of descending from a common ancestral sequence). Pangenomes are emerging as a replacement for linear reference assemblies to help mitigate these problems [[22](#), [77](#), [9](#)]. They can particularly improve genotyping of structural variants[[41](#)].

Pangenomes are frequently formulated as sequence graphs[[33](#)]—mathematical graphs that represent the homology relationships between multiple sequences. Several algorithms have

been developed for mapping sequences to sequence graphs. None has yet made mapping the short sequencing reads from widely used DNA sequencers, such as those made by Illumina, to a structurally complex pangenome a practical option for large-scale applications. The original VG-MAP algorithm[37] maps to complex sequence graphs that contain cycles produced by duplications and complex genomic rearrangements[37]. However, VG-MAP is at least an order of magnitude slower than popular linear genome mappers that have comparable accuracy. Given that mapping is frequently a bottleneck in genome analysis, the cost of VG-MAP has proven prohibitive. Other pangenome mappers have different capabilities and limitations. Some are faster but are limited to acyclic graphs that contain variation at relatively low density[47], and some can map to arbitrary sequence graphs but are designed for long reads[72]. Other tools are not open source and are thus unavailable for general testing and customization[70, 42], and some additionally cannot run on commodity computing environments[42].

## 4.5 Results

### 4.5.1 Giraffe: Fast, haplotype-aware pangenome mapping

When a sequence graph reference[22] (Figure 4.8) is substituted for the traditional linear reference (Figure 4.2 A), it can reduce reference allele bias by including more alleles[37]. However, it also expands the size of the alignment search space from a few linear chromosome strings to a combinatorially large number of paths in the graph. This has made our previous graph mappers slower than linear mappers[37]. Giraffe solves this problem by considering the paths that are observed in individuals' genomes: the reference haplotypes. We use the two

haplotypes (one from each parent) that each individual has in their genome and trace them as paths through the sequence graph. The graph describes which positions in the haplotypes are equivalent, whereas the haplotypes describe the subset of the possible paths in the graph to consider. Giraffe uses a graph Burrows-Wheeler transform (GBWT) index[79] to store and query a graph’s haplotypes efficiently.

Giraffe’s strategy of aligning to haplotype paths has two key benefits. First, it prioritizes alignments that are consistent with known sequences, thereby avoiding combinations of alleles that are biologically unlikely. Second, it reduces the size of the problem by limiting the sequence space to which the reads could be aligned. This deals effectively with complex graph regions where most paths represent rare or nonexistent sequences.

We designed Giraffe to minimize the amount of gapped alignment that is performed. Computing gapped alignments, in which sequences are allowed to gain or lose bases relative to each other, is much more expensive than gapless alignment because it requires pairwise dynamic programming algorithms. Most Illumina sequencing errors are substitutions[74], and common true indels relative to the traditional linear reference should already be present in the haplotypes; therefore, almost all reads will have a gapless alignment to some stored haplotype. Hence, we try to align each read without gaps before resorting to dynamic programming.

Giraffe follows the common seed-and-extend approach used by most existing mappers (Section 4.9.1). In this framework, short seed matches between a sequencing read and a genomic reference are found with minimal work, and then only good seeds are extended into mappings of the entire read [50, 53, 55]. A visual overview of Giraffe’s operation is given in (Figure 4.2, B to F). The Giraffe algorithm uses several heuristics for prioritizing alignments.

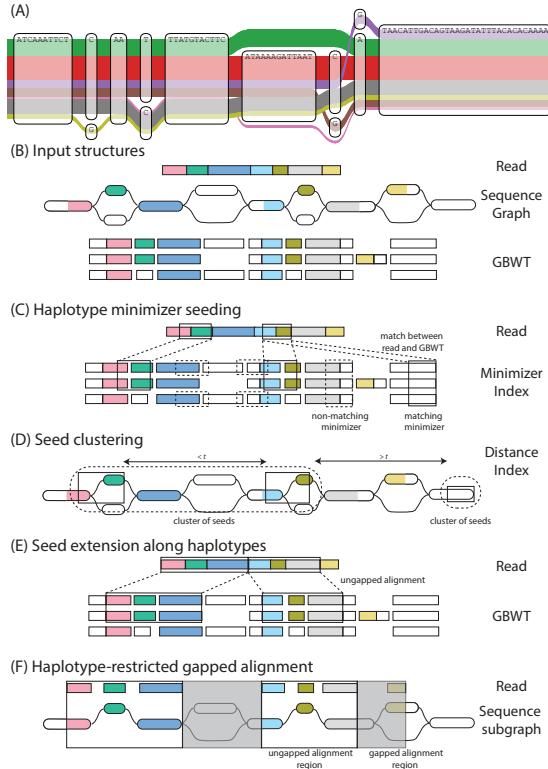
These heuristics are configurable, and we present two presets: default Giraffe (written as just “Giraffe”) balances speed and accuracy, and fast Giraffe optimizes for speed at the expense of some accuracy.

#### 4.5.2 Pangenome references for evaluation

To evaluate Giraffe, we built two human genome reference graphs based on the GRCh38 reference assembly. One (the 1000GP graph) contained mostly small [ $\leq 50$  base pairs (bp)] variants from the 1000 Genomes Project [5]. The other (the HGSVC graph) contained entirely SVs ( $\geq 50$  bp) from the Human Genome Structural Variant Consortium[15]. The 1000GP graph contained data from 2503 individuals, with one (NA19239) held out for benchmarking. It was built from 76,749,431 SNVs; 3,177,111 small indels ( $\leq 50$  bp); and 181 larger SVs ( $/geq 50$  bp). The HGSVC graph contained data from three individuals sequenced with long reads: HG00514, HG00733, and NA19240. The HGSVC graph contained 78,106 larger SVs ( $\leq 50$  bp). Both graphs are available for reuse (see Data and materials availability in the Acknowledgments).

#### 4.5.3 Giraffe and VG-MAP map accurately to human pangenomes

We evaluated Giraffe for mapping human data by simulating paired-end reads for two individuals (Section 4.9.2.4): NA19240, who has available genotypes for the HGSVC variants [15], and NA19239, who has available genotypes for the 1000GP variants[5]. Simulated read sets were mapped using Giraffe and competing tools (Section 4.9.2.5). We examined the accuracy of single- and paired-end mapping (Figure 4.3). We looked at a variety of input read

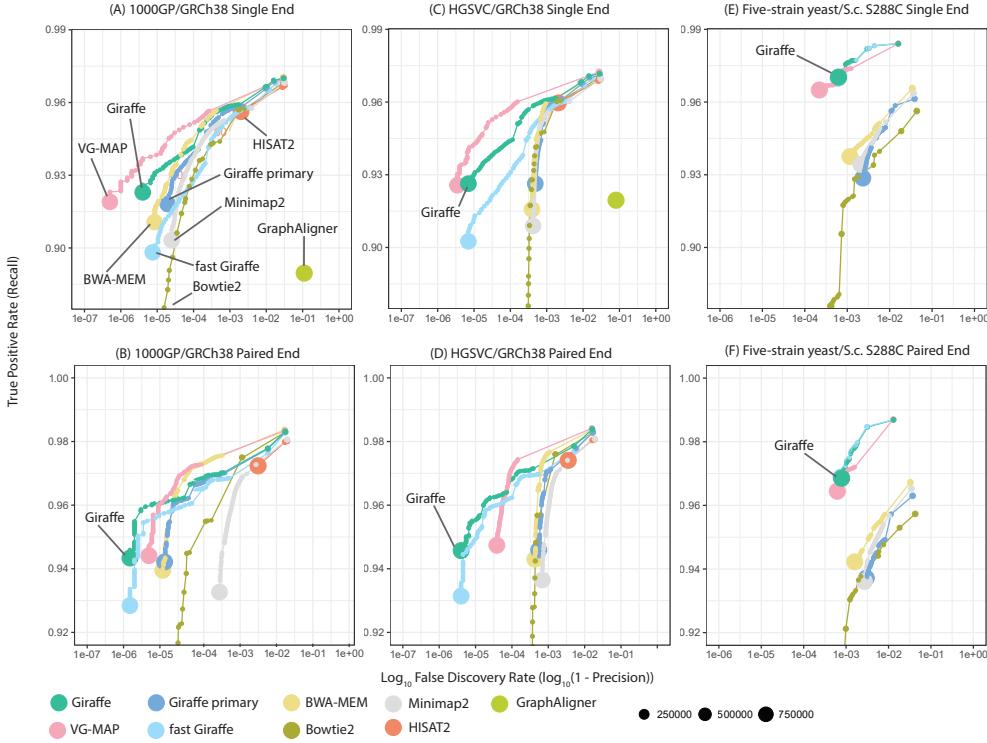


**Figure 4.2: Haplotype mapping.** (A) A region of the CASP12 gene in the 1000GP graph, illustrating complex local variation. The observed haplotypes (the colored ribbons of width log-proportional to population frequency) represent only a subset of the possible paths through the graph. (B to F) An overview of Giraffe. Input structures are shown in (B): Giraffe takes as input each read to map, the sequence graph reference to map against, and the GBWT of known haplotypes to restrict to. The input read is represented as a series of colored rectangles. The haplotype sequences in the GBWT are similarly represented as series of rectangles, split according to the nodes they correspond to in the sequence graph. Nodes in the sequence graph and haplotypes in the GBWT are colored according to homology with the read. Haplotype minimizer seeding is shown in (C): Seeds are identified using an index of minimizers (subsets of sequences of specified length  $k$ )<sup>[73]</sup> over the sequences of all the GBWT haplotypes. A matching minimizer between the read and the GBWT haplotypes constitutes a seed. The minimizers (black boxes) in the read are enumerated and the matching minimizers in the haplotypes are identified using the minimizer index. Seed clustering is shown in (D): Minimizer instances in the graph are clustered by the minimum graph distance ( $t$ , measured in nucleotides) between them<sup>[16]</sup>. Seed extension along haplotypes is shown in (E): Minimizers in high-scoring clusters are extended linearly to form maximal gapless local alignments. Haplotype-restricted gapped alignment is shown in (F): Giraffe is designed on the assumption that for most reads, it will be possible to gaplessly extend seed alignments all the way to the ends of the read, allowing the algorithm to stop at the previous step. However, any remaining gaps in the alignment between read and graph are resolved by gapped alignment in this final step.

sets and evaluated the calibration of reported mapping quality, which is a standard measure of mapping uncertainty (Figures 4.10, 4.11, 4.12, 4.13, 4.14, 4.15 and Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6). Relative to other tools, at the highest reported mapping quality, VG-MAP and default Giraffe consistently have either higher precision or higher recall across all simulated read technologies and graphs. Their performance is generally similar. Relative to the linear mappers, the Giraffe and VG-MAP lead is larger for the HGSVC graph (Figure 4.3, C and D) than for the 1000GP graph (Figure 4.3, A and B). This suggests that the gains from using a genome graph are higher when the graph facilitates alignment of genomic sequences from the sample that differ greatly from the linear reference.

#### 4.5.4 Haplotype sampling improves read mapping

Having rare variants or errors in the graph and haplotypes may reduce mapping accuracy by creating opportunities for false-positive mappings[66]. Mapping reads to regions with many distinct local haplotypes can also be slow. Additionally, Giraffe needs a mechanism to synthesize haplotypes for graph components where no haplotype variation is known. To overcome these issues, Giraffe includes mechanisms for creating synthetic haplotype paths. When real haplotypes are available, these synthetic haplotype paths represent local haplotype variation sampled according to haplotype frequency, and we call the result a sampled GBWT (Section 4.9.1.3). When no haplotypes are available, we call the result a path cover GBWT. In this case, the synthetic haplotypes represent random walks through the graph. We evaluated the effects of running our mapping evaluations with sampled and path cover GBWTs (Figure 4.16 and Tables 4.8 and 4.7; Section 4.9.1.4). The mapping benefit of sampling more



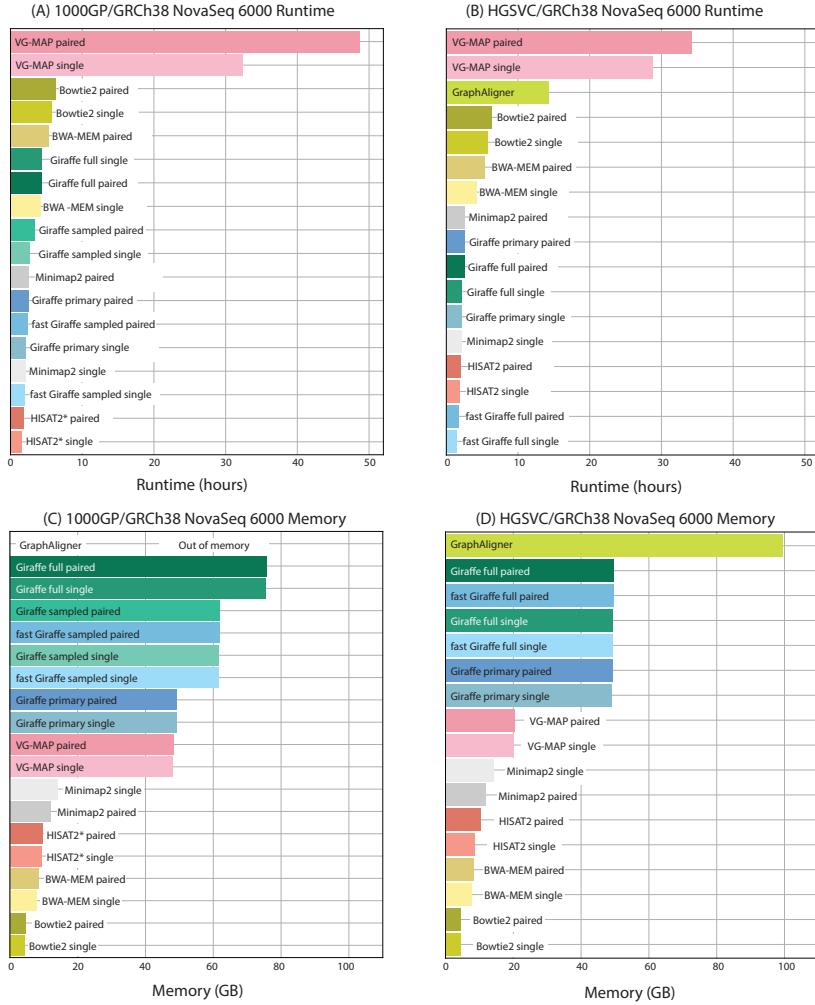
**Figure 4.3: Simulated read mapping** (A to F) Each panel shows recall versus false discovery rate (or 1 minus precision) for a simulated read-mapping experiment, comparing Giraffe with linear genome mappers (BWA-MEM, Bowtie2, and Minimap2) and other genome graph mappers (VG-MAP, GraphAligner, and HISAT2). Reads were simulated to match  $\sim 150$ -bp Illumina NovaSeq (for human) or HiSeq 2500 (for yeast) reads, either as single-end reads [(A) to (C)] or as paired-end reads [(D) to (F)] (Section 4.9.2.4). Results for each mapper are shown stratified by reported read-mapping quality; the size of each point represents the log-scaled number of reads with the corresponding mapping quality. Three different mapping scenarios are assessed: [(A) and (D)] Comparing mapping to a graph derived from the 1000GP data to mapping to the linear reference genome assembly upon which it is based (GRCh38); [(B) and (E)] comparing mapping to a graph containing larger structural variants from the HGSVC project to mapping to the GRCh38 assembly upon which it is based; and [(C) and (F)] comparing mapping to a multiple sequence alignment-based yeast graph to mapping to the single *S.c. S288C* linear reference, for reads from the DBVPG6044 strain. For mapping with Giraffe, we used the full GBWT that contains six haplotypes to map to the HGSVC graph and the 64-haplotype sampled GBWT to map to the 1000GP graph. “Giraffe primary” represents mapping with Giraffe to the linear reference.

haplotypes plateaued at 64 haplotypes for the 1000GP graph (which contains around 5000 haplotypes), with higher accuracy than that achieved by mapping to the full haplotype set. We used the HGSVC graph (which contains just six haplotypes) for an experiment on generating path covers without known haplotypes. Path covers alone did not outperform the full underlying haplotype set for the HGSVC graph but came close to matching its performance. We selected the 64-haplotype sampled GBWT for the 1000GP graph and the full GBWT for the HGSVC graph as the best-performing GBWTs, which we use in the rest of the analysis.

#### 4.5.5 Giraffe improves pangenome mapping speed

We measured the runtime (Figure 4.4, A and B) and memory usage (Figure 4.4, C and D) of Giraffe and competing tools when mapping real reads (Section 4.9.2.7). Giraffe was more than an order of magnitude faster than VG-MAP in all conditions. It was also faster at aligning to human graphs than Bowtie2 or BWA-MEM were at aligning to the corresponding linear reference. For the 1000GP graph, using the 64-haplotype sampled GBWT for mapping instead of the full  $\sim$ 5000-haplotype GBWT was much faster in every case. HISAT2 and fast Giraffe were both about equally fast and were both faster than all other mappers.

Because of the in-memory indexes it uses, Giraffe’s memory consumption is higher than the other mappers, except for GraphAligner. However, it can map to the 1000GP graph with the full GBWT in  $\sim$ 80 gigabytes (GB) of memory—an amount readily available on compute cluster nodes (Figure 4.4, C and D).



**Figure 4.4: Runtime and memory usage** (A to D) Total runtime [(A) and (B)] and peak memory use [(C) and (D)] for mapping  $\sim$ 600 million NovaSeq 6000 reads using 16 threads. Reads were mapped [(A) and (C)] to the 1000GP derived graph or (for linear mappers) the GRCh38 assembly and [(B) and (D)] to the HGSVC graph or GRCh38 reference, respectively. For HISAT2\*, results are shown for the subset 1000GP graph. “Giraffe full” refers to mapping using the full GBWT of all haplotypes. “Giraffe sampled” refers to mapping using the 64-haplotype sampled GBWT.

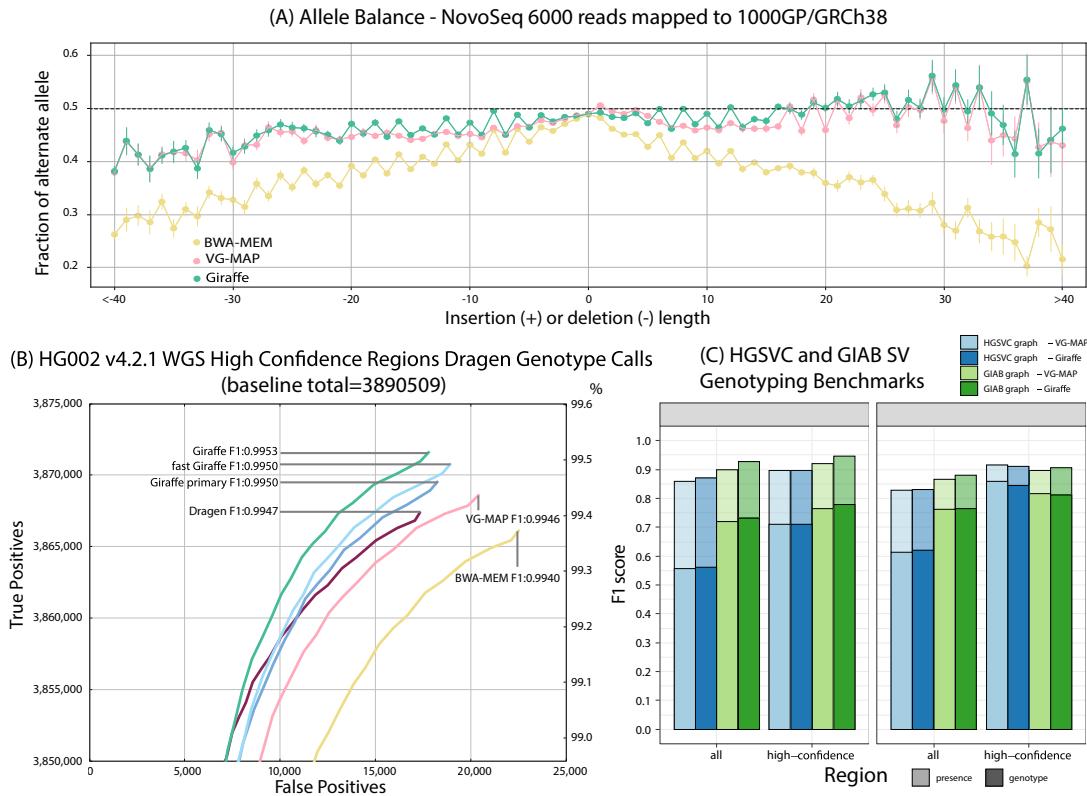
#### 4.5.6 Giraffe reduces allele mapping bias

We assessed Giraffe’s reference bias (Section 4.9.3). We expected Giraffe to be able to use the extra variation information contained in the graph reference to achieve a lower level of bias than a linear mapper. For variants that were heterozygous in NA19239, we found the fraction of reads supporting alternate versus reference alleles at each indel length (Figure 4.5 A). Giraffe and VG-MAP both show less bias toward the reference allele than a linear mapper, and this difference becomes more pronounced as indel length increases, particularly for larger insertions.

#### 4.5.7 Giraffe genotyping outperforms best practices

We used Illumina’s Dragen platform [42] to genotype SNVs and short indels using Giraffe mappings to the 1000GP graph, projected onto the linear reference assembly. We compared these results with results using competing graph and linear reference mappers (Section 4.9.4). No training or optimization was performed for any of the mappings other than those performed by default by Dragen itself. We evaluated the calls using the Genome in a Bottle (GIAB) v4.2.1 HG002 high-confidence variant-calling benchmark [89].

Out of the examined pipelines, Giraffe mappings to the 1000GP graph produce the highest overall F1 score (harmonic mean of precision and recall) at 0.9953 (Figure 4.5B and Tables 4.9, 4.10). Similar but uniformly higher results were found with higher-coverage, 250-bp reads (Figure 4.17, Tables 4.11, 4.12). Although one would expect longer reads and higher coverage to produce better variant calls, with all else being equal, Giraffe has a slightly higher F1 score with the 150-bp read set (0.9953) than BWA-MEM with the higher coverage 250-bp read



**Figure 4.5: Evaluating Giraffe for genotyping.** (A) The fraction of alternate alleles in reads detected for heterozygous variants in NA19239. Reads were mapped to the 1000GP graph with Giraffe and VG-MAP and to GRCh38 with BWA-MEM, and the fraction of reads supporting reference or alternate alleles was found for each indel length. (B) Assessing true-positive and false-positive genotypes made using the Dragen genotyper with mappings from Giraffe and other mappers. The line labeled Dragen represents the mapper included with the Dragen system itself. (C) Comparing Giraffe with VG-MAP for typing large insertions and deletions. “Presence” (lighter bars) evaluates the detection of SVs without regard to genotype; “genotype” (darker bars) requires the SV to be detected and its genotype to agree with the truth genotype. The y axis shows the F1 score. For the HGSVC benchmark, we define high-confidence regions as regions not overlapping simple repeats and segmental duplications. For the GIAB benchmark, we use the set high-confidence regions provided by GIAB.

set (0.9952). Restricting comparison only to confident regions that overlap variant calls from the 1000GP variants used in graph construction, Giraffe has the highest F1 score at 0.9995 relative to the other methods (Table 4.13 and Figure 4.18). Perhaps surprisingly, Giraffe maintains the highest F1 score (0.9528) when performing the converse analysis, restricting the comparison to confident regions that do not overlap 1000GP variant calls (Table 4.14 and Figure 4.19).

DeepVariant is a highly accurate genotyping tool that requires training [65]. We trained DeepVariant to use Giraffe mappings and evaluated it on the held-out sample HG003 (see 4.7.1.5). We compared it with the Dragen pipelines tested and DeepVariant using BWA-MEM with the BWA-MEM trained model that the developers provide. The Giraffe-DeepVariant pipeline (F1: 0.9965) outperforms all other tested pipelines (Tables 4.15, 4.16 and Figure 4.20).

Previously, when we used VG-MAP to map reads to SV pangenomes, we found it to perform better than other methods for SV genotyping [41]. We replicated that evaluation on the HGSVC and GIAB datasets [94, 15] to confirm that the quality of the SV genotypes from Giraffe was competitive (Section 4.9.5). We observed similar SV genotyping accuracy across SV types, genomic regions, and datasets (Figure 4.5 C). Of note, GraphTyper [31], which was published after our earlier benchmarking analysis [41], was also compared with vg as a variant caller but showed lower genotyping performance across SV types, genomic regions, and datasets (Figure 4.21).

#### 4.5.8 Giraffe generalizes beyond human

We assessed Giraffe’s performance mapping to a yeast pangenome for five strains of the *Saccharomyces cerevisiae* and *Saccharomyces paradoxus* yeasts (Section 4.7.1.6). This

graph was substantially different from the human graphs. It proved challenging because it contains the cycles and duplications typical of graphs generated from genome-wide alignments of more divergent sequences. Using a graph decomposition technique [63], we find it contains 1,459,769 variant sites, four times the density of variation in the 1000GP graph. Ninety of these sites are complex, meaning that they are not directed, not acyclic, or not free of internal source and sink nodes.

Mapping accuracy results for reads from the held-out *S. cerevisiae* DBVPG6044 strain are displayed in Figure 4.3, E and F, for the single-end and paired-end reads, respectively. Speed results for mapping real reads are presented in (Figure 4.22). Neither HISAT2 nor GraphAligner could map reads to the yeast graph; in the case of HISAT2, this was because it cannot map to graphs containing cycles. Giraffe is 28 times faster for paired-end mapping than the only other tool that could map to this graph, VG-MAP, while achieving similar accuracy. Both graph mappers are much more accurate than the linear reference methods. Moreover, the gap between the graph methods and the linear reference is even larger on this graph than in the HGSVC graph (Figure 4.3, C and D). This lends further support to the hypothesis that graph-mapping methods have the most benefit when facilitating alignment of genomic sequences that differ greatly from the reference, such as the sibling-subspecies-scale differences represented in the yeast graph.

#### 4.5.9 Genotyping the SVs of the 5202 samples

Building on our previous work to genotype SVs [41], we demonstrate the value of Giraffe by performing population-scale genotyping of an expanded compendium of SVs in

large cohorts of samples sequenced with short reads. We built a comprehensive pangenome containing SVs that combines variants from three catalogs of SVs that were discovered using long-read sequencing [94, 15, 8]. The combined catalog represents 16 samples from diverse human populations and is estimated to cover most of the common insertions and deletions in the human population [8]. Near-duplicate versions of variants (i.e., SVs with slightly different breakpoints) are often present within and across SV catalogs. A naïve integration of all these variants can lead to redundancy in the graph that can affect read mapping and variant genotyping. We remapped sequencing data and integrated variants iteratively into the graph to progressively build a nonredundant, compact SV graph (Section 4.9.6). The final SV graph was constructed from 123,785 SVs from the original catalogs: 53,663 deletions and 70,122 insertions. Overall, the graph contained 26.2 Mbp of nonreference sequences in the form of insertions. Using a graph decomposition [63], we identified 228,405 subgraphs that represent variant sites. Some of these correspond to smaller variants nested inside larger ones. After combining these cases, there were 96,644 non-nested, nonoverlapping SV subgraphs.

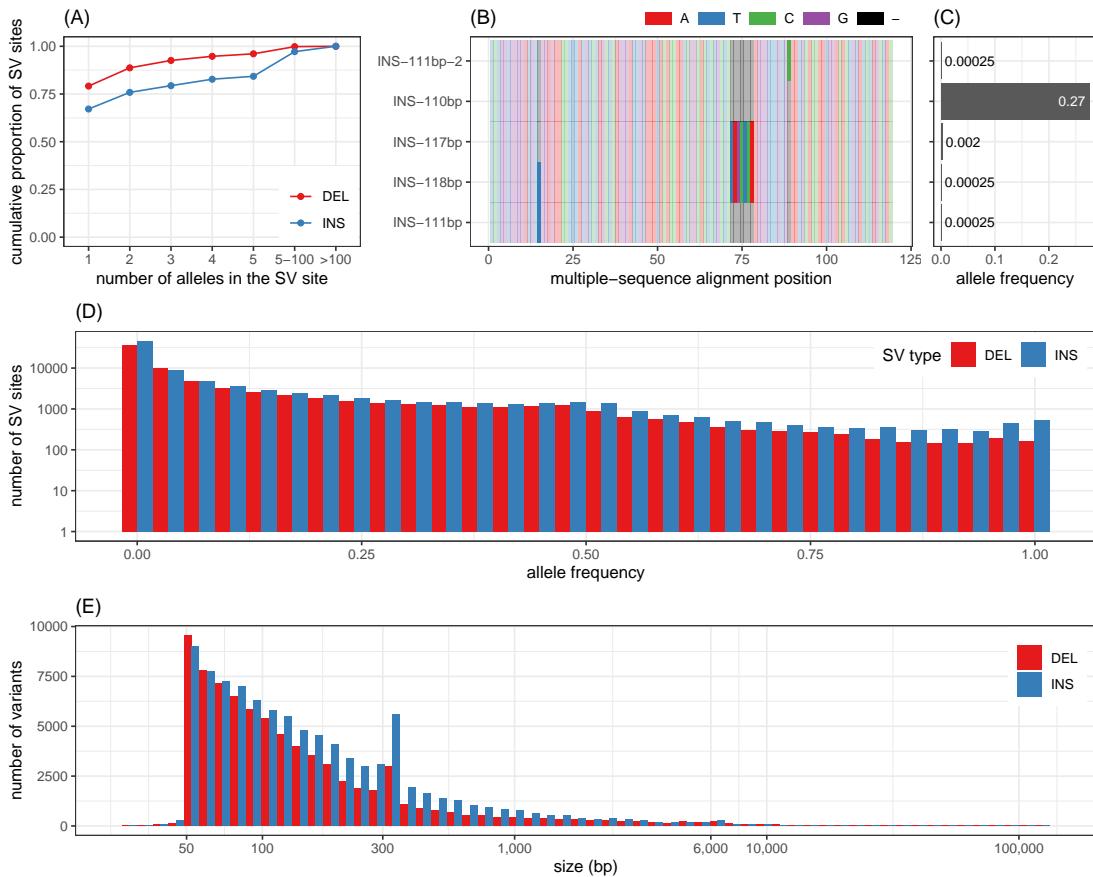
Compared with Hickey et al. [41], we used a graph containing more SVs and a more recent version of the vg toolkit (see Data and materials availability in the Acknowledgments), including the Giraffe mapper presented here. Our SV genotypes were as accurate as those in [41], if not more so (Figure 4.24A-C). Thanks to Giraffe and improvements in the variant calling approach, the genotyping workflow used about 12 times less compute on a sample sequenced at about 20 $\times$  coverage.

SV genotyping was run using the NHLBI BioData Catalyst ecosystem [62] (Figure 4.24D). We genotyped samples from the Multi-Ethnic Study of Atherosclerosis (MESA) co-

hort within the Trans-Omics for Precision Medicine (TOPMed) program. The MESA cohort is a longitudinal cohort study consisting of 6814 participants at baseline (between the years 2000 and 2002). Participants were ascertained from six sites in the United States and identified themselves as “Spanish/Hispanic/Latino” (22% at baseline) and/or “African American or Black” (28%), “Chinese” (12%), and “Caucasian or White” (39%) [13]. Two-thousand samples from the MESA cohort [13] were selected, using a criterion to maximize the sample diversity (Section 4.9.7.3). Using the graph described above and fast Giraffe, it took around 4 days to genotype 2000 samples from the MESA cohort. We used the same workflow to genotype the 3202 diverse samples from the high-coverage 1000GP dataset [14] in around 6 days. On average, genotyping a sample took 194.4 central processing unit (CPU)-hours of compute and cost between 1.11 and 1.56 (Figure 4.24D, Tables 4.17 and 4.21). The sequencing data were down-sampled in advance to  $\sim$ 20 $\times$  coverage to reduce compute costs. Benchmarking indicates that this down-sampling has a minimal impact on the genotyping accuracy (Figure 4.25).

#### 4.5.10 Diverse, clustered SVs

Our SV graph construction approach preserves multiple alleles cataloged at a given SV site and decomposes them into a parsimonious joint representation. In general, these SV representations require more alleles per site than is common for small variants. For example, there may be SNVs and indels within the sequence of an insertion or around a deletion’s breakpoints, or copy-number changes in variable number tandem repeat (VNTR) regions. The existence of these potentially recombining subvariants implies the possibility of previously unidentified alleles.



**Figure 4.6: SVs in the MESA cohort.** (A) Cumulative proportion of SV sites depending on the maximum number of alleles (x axis) in the site. DEL, deletion; INS, insertion. (B and C) Illustration of an insertion site with five alleles. The alleles differ by three nested indels as shown by the multiple sequence alignment of the inserted sequences represented in (B). Only one allele is frequent in the population (allele frequency of 0.27), as highlighted in (C). (D) Allele frequency distribution of the major allele for each SV site. The y axis, showing the number of SVs, is log-scaled. (E) Size distribution of the major allele for each SV site.

We genotyped a total of about 1.7 million alleles clustered in 167,858 SV sites across the 2000 MESA samples. In most SV sites, we only observed one or a few alleles ( $\sim 90\%$  of SV sites with five or fewer alleles; Figure 4.6 A). Additionally, most of the SV sites ( $\sim 151,000, 90\%$ ) contained SV alleles that differed by only small variants (Section 4.9.7.5), whereas the rest of the sites showed size variation from polymorphic VNTR regions (Figure 4.26A). Examples of SV sites that illustrate these different profiles are given in Figure 4.6 B and C, and Figure 4.27. Figure 4.6, D and E, shows the size and frequency distributions of the most common allele at each site. SVs spanned the size spectrum (50 bp up to 125 kbp), with 89.8% shorter than 500 bp. For 84% of the SVs, simple repeats or low-complexity regions overlapped at least 50% of the SV region. Hence, the SVs genotyped in this study match the original SVs discovered in long-read sequencing studies [94, 15, 8] in terms of number, size distribution, and sequence context.

We observed similar patterns in the 1000 Genomes Project dataset. We identified 1.9 million alleles clustered in 167,188 SV sites, with a size and frequency distribution similar to that of the MESA cohort (Figure 4.28). The 1000 Genomes Project dataset also provided 602 trios that we used to estimate the quality of our genotypes. First, we computed the rate of Mendelian error, which was 5.2% for deletions and 4.7% for insertions when considering all variants. This error decreased as the confidence in the genotype increased. For example, the Mendelian error dropped to 2.1 and 2.5% for the  $\sim 70\%$  of deletions and insertions, respectively, with the highest genotype qualities (Figure 4.29). The most common error by far occurred when a heterozygous variant was predicted in the offspring but both parents were predicted homozygous for the reference allele (Table 4.18). The transmission rate of heterozygous alleles

was close to the expected 50%: 40 to 47% for deletions and 43 to 49% for insertions (Figure 4.29B).

#### 4.5.11 Comprehensive SV frequency estimates

The genotyped SVs were originally discovered with long-read sequencing technology, and many are absent from the population scale SV catalogs that could provide frequency information. Of the SV sites genotyped using our pangenome approach, 93% are missing from the 1000 Genomes Project SV catalog [84], and 67% were missing from the Genome Aggregation Database (gnomAD)–SV catalog [38] (Table 4.19). This is consistent with the amount of previously unidentified structural variation described in the three studies from which our SV graph is derived [94, 15, 8]. Our results provide frequency estimates across a large and diverse cohort for these SVs.

The frequency distribution resembled the allele frequency distributions in the 1000 Genomes Project SV and gnomAD-SV catalogs (Figure 4.30A). The frequencies of the subset of variants present in both our catalog and the mentioned public catalogs were largely concordant (Figure 4.30B-C). Our frequency distribution looks rather different than that of SVPOP [8]. However, we note that SVPOP’s frequency distribution is markedly different than the 1000 Genomes Project and gnomAD-SV (Figure 4.30A) and has very different frequency estimates on matched variants (Figure 4.30D,F).

#### 4.5.12 Fine-tuning SVs with frequencies

SVs in the input catalogs may contain errors. When multiple alleles co-occur at an SV site, we often observed that one allele was frequently present in the cohort, whereas other similar alleles were not (Figure 4.6 B and C). The other alleles at these sites are either rare or erroneous. In either case, it is useful to identify the major alleles. In 7520 SV sites, only one allele was called in more than 1% of the population, whereas other alleles from the original catalogs were not. Further, the major allele was at least three times more frequent than the second most frequent allele in 6175 of these sites (Figure 4.26B). As a quality control, we verified that these alleles were more likely to match exactly with the alleles in the GIAB truth set [94], which is the SV catalog with the highest base-level confidence [permutation p  $\leq$  0.0001; Section 4.9.7.8]. Our results thus help fine-tune the sequence resolution of these SVs. More generally, our results identify one major allele for 39,699 multiallelic SV sites.

#### 4.5.13 SV frequency population signatures

Principal components analysis (PCA) of the allele counts at the 166,959 SV sites in the MESA cohort produces a low-dimensional embedding of the samples. This embedding appears similar to the TOPMed consortium's PCA of SNV genotype data from all samples (Pearson correlation of 0.96 to 0.99 for the top three components; Figure 4.31). This result is expected and provides confirmatory support for the accuracy of our SV genotypes.

We clustered samples with PCA, taking each cluster to be a population (Section 4.9.7.6). Allele frequencies vary across these populations for thousands of SV sites (Figure 4.32A-C). For example, we found 21,069 SV sites with strong intercluster frequency patterns,

defined by a frequency in any population differing by more than 10% from the median frequency across all populations (Figure 4.32D). The existence of SVs with different frequencies across populations supports the need to develop and test genomic tools and references across multiple populations.

Because there is a risk of circularity when using the same genotype data to define populations and look for patterns across them, we replicated these observations in the high-coverage 1000 Genomes Project dataset [14]. Here, again, the PCA of the allele counts organized the samples in a way consistent with the known history of the 1000 Genomes Project “superpopulation” groups (Figure 4.33). In this analysis, we found 25,960 SV sites with strong inter-superpopulation frequency patterns, defined as for the MESA analysis, but with the 1000 Genomes superpopulations as the sample categories (Figure 4.34). As a comparison, when the samples were randomly grouped into superpopulations, we observed only 14 SV sites with strong intergroup frequency patterns (Section 4.9.7.6). More than 17,000 SV sites with strong inter-superpopulation frequency patterns were enriched or depleted in the African Ancestry (AFR) superpopulation, followed by about 10,000 sites enriched or depleted in the East Asian Ancestry (EAS) superpopulation.

As an example of a newly annotated variant, a deletion of the RAMACL gene was genotyped with frequency 46.6% in the AFR super population, 4% in American Ancestry (AMR), and less than 1% in other superpopulations. This deletion is not present in the 1000 Genomes Project SV catalog and was unresolved in version two of the gnomAD-SV catalog. It has been curated in gnomAD-SV v2.1 and shows similar population patterns there to what we found in our reanalysis of the 1000 Genomes Project dataset. Such variants could be falsely

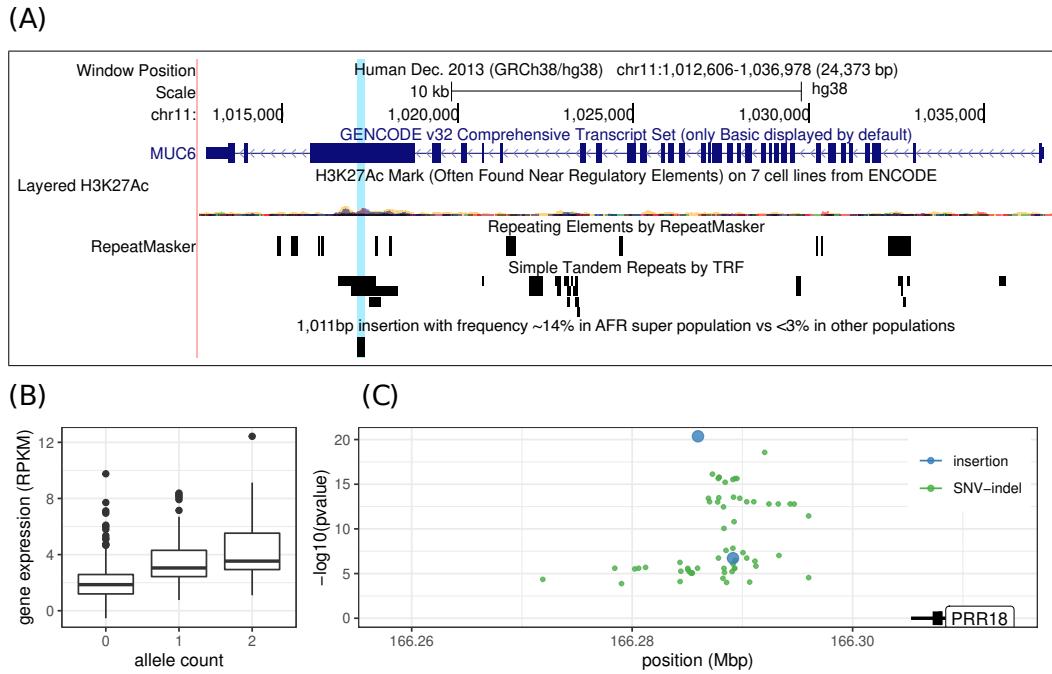
identified as putatively pathogenic if analyzed only in European-ancestry populations where the frequency is low.

In addition, our approach is often capable of genotyping repeat-rich variants, such as short tandem repeats that vary in length. For example, a 1-kbp expansion of an exonic VNTR in MUC6 with a frequency of 14% in the AFR superpopulation was observed only rarely outside of it: 2.3% in AMR and < 1% in other superpopulations (Figure 4.7 A). This repeat expansion is absent from gnomAD-SV and the SV catalog from the 1000 Genomes Project, despite its observed frequency.

#### 4.5.14 SVs, genes, and expression

In the MESA and 1000 Genomes Project datasets, 1563 and 1603 SVs overlapped coding regions of 408 and 380 protein-coding genes, respectively. When including promoters, introns, and untranslated regions, each dataset had overlaps between at least 78,290 SVs and 7641 protein-coding genes. Of these SVs, 10,640 show strong inter-superpopulation frequency patterns in the 1000 Genomes Project dataset (see Figure 4.7 A).

We searched for associations between SVs and gene expression across 445 samples from the 1000 Genomes Project that have been RNA sequenced by the Genetic European Variation in Disease (GEUVADIS) consortium [87]. These samples span four European-ancestry populations [Utah residents (CEPH) with Northern and Western European ancestry (CEU), Finnish in Finland (FIN), British in England and Scotland (GBR), and Toscani in Italy (TSI)], and the Yoruba in Ibadan, Nigeria (YRI) population [87]. A pooled analysis identified 2761 expression quantitative trait loci (eQTLs) across 1270 genes [false discovery rate of 1%; Section



**Figure 4.7: Population-specific SVs and SV-eQTLs in the 1000 Genomes Project dataset.** (A) Example of an insertion at appreciable frequency (~ 14%) in the AFR superpopulation that is rare (< 3%) in the other superpopulations. The variant is a 1011-bp expansion of a VNTR in the coding sequence of the MUC6 gene. chr11, chromosome 11; TRF, Tandem Repeats Finder. (B and C) Association between a 10,083-bp insertion overlapping a predicted enhancer and the gene expression of the PRR18 gene. Each allele is associated with an increase in gene expression, as shown in (B). The position of significant eQTLs (SNV-indels in green, insertions in blue) is shown in (C). All the eQTLs are in the intergenic region downstream of the PRR18 gene. The y axis represents the significance of the association, with the top eQTL being the highest point. Of note, the lead eQTL (the 10,083-bp insertion) overlaps a region predicted to be an enhancer by ENCODE. In (B), boxes represent the median and quartiles; whiskers extend from the box up to 1.5 times the interquartile range.

[4.9.7.10](#)]. Of those genes, 878 are protein-coding genes. We note that 58% of the SV-eQTLs are located within simple repeats or low-complexity regions. The distribution of the p values across all tests showed the expected patterns for genome-wide association studies (Figure [4.35](#)).

Genes with eQTLs, or eGenes, were enriched in gene families involved in immunity, as previously observed [35], but we also found significant enrichments in other families (Table [4.20](#)). For example, 3 of the 10 genes in the anoctamins family have SV-eQTLs (adjusted p = 0.0006). This gene family is involved in the regulation of multiple processes, including neuronal cell excitability, and mutations in some of its members have been linked to neurologic disorders [10]. Other families enriched included the survival motor neuron (SMN) complex family (3 out of 10 genes with an SV-eQTL, adjusted p = 0.0012) and aldehyde dehydrogenases genes (3 out of 19 genes with an SV-eQTL, adjusted p = 0.008). As expected, SV-eQTLs were strongly enriched in coding, intronic, promoter, untranslated, and regulatory regions (Figure [4.36](#)). Interestingly, SVs associated with decreased gene expression drove most of the enrichment in coding regions. Separate analysis of the four European-ancestry populations together and the YRI population alone identified, respectively, 44 and 139 SVs where an association with the expression of protein-coding genes was detected only in the smaller analysis (Section [4.9.7.10](#)). As expected, a number of these population-specific SV-eQTLs had shown strong inter-superpopulation frequency patterns (see above).

Finally, we performed a joint analysis with available SNV and indel calls. Like previous studies [18, 29], we found that the lead eQTLs (the strongest association for a gene) are enriched in SVs (permutation p = 0.022). For example, only 0.5% of the variants tested were SVs, but SVs were the lead eQTL in 5.9% of the genes that had both SV and SNV-indel eQTLs.

We did not observe a difference in relative effect size of SV-eQTLs compared with SNV-indel eQTLs, but we noticed that SV-eQTLs were fourfold enriched in the genes with the highest expression (permutation  $p = 0.004$ ; Figure 4.37). Figure 4.7, B and C, and Figure 4.38 show two examples where the SV-eQTL is the strongest association: a 10,083-bp insertion associated with an increased expression of the PRR18 gene and a 5405-bp deletion associated with a reduced expression of the SLC44A5 gene. In addition, 39 genes had SV-eQTLs but no SNV-indel eQTLs (Table 4.20). These results show that the SV genotypes produced here can be used to test for phenotypic association.

## 4.6 Discussion

Pangenome references hold great potential as a replacement for standard linear reference genomes. They can represent diverse collections of human genomes, and they have been shown to reduce the bias that arises from using a linear reference [37]. However, because of the appreciable complexity of the task, previous methods for mapping to pangenomes have been slow or not clearly better than comparable methods for linear genomes. By contrast, Giraffe can map to pangenome graphs consisting of thousands of aligned haplotypes, potentially with complex topologies, with accuracy comparable to that of the best previously published tools and speed surpassing linear reference mappers. Further, we have demonstrated that its mappings can improve genotyping.

Pangenome exchange formats have been coevolving alongside pangenome methods. Giraffe is designed to meet and solidify these emerging standards while also interfacing with the

broader genomics ecosystem. The graphical fragment assembly (GFA) format for representing pangenome graphs has increasing tool support, including by vg [72, 56, 48, 54, 90, 67]. In addition, Giraffe can output the graphical mapping format (GAF) read-to-pangenome graph alignment format proposed by Li et al. [56] and supported by other pangenome mappers [72]. Giraffe also supports backward compatibility to linear references by allowing mappings to be projected onto an embedded linear reference genome and output in standard formats. The state-of-the-art SNV and short-indel genotyping results described in this study demonstrate the value of this support. These necessary technical advances are starting to nucleate an interoperable tool ecosystem for pangenomics.

For SVs, and for particularly large insertions, we and others have shown that the benefits of pangenomes for genotyping are not merely incremental but transformative [41, 56, 17]. Our approach allowed us to identify duplicate SVs, to refine the canonical definitions of SVs, and to establish the frequencies of these SVs in diverse human populations. Complementing previous surveys of SVs in diverse human populations [84, 29, 83], we demonstrate that many of the previously unidentified SVs studied here are also differentially distributed across human populations. This frequency information could be used, among other applications, for prioritizing variants to investigate for genomic medicine because variants common anywhere are unlikely to be pathogenic.

We expect accurate and unbiased SV genotyping to be one of the most impactful contributions of pangenomics. Among other applications, this contribution will enable more links from SVs to disease traits and other phenotypes to be identified. For example, we were able to detect thousands of associations between SVs and gene expression. Ebert et al. [29] recently

performed a similar analysis using the same RNA sequencing (RNA-seq) dataset from the GEU-VADIS consortium (complemented with 34 new deep RNA-seq experiments) and genotypes for SVs discovered in 32 haplotype-resolved genomes. Although we did not use this new sequencing data and SV catalog, we found a similar number of SV-eQTLs with our pangenomic approach [2761 SV-eQTLs and 1270 eGenes in this study; 2109 SV-eQTLs and 1526 eGenes in Ebert et al. [29]].

Soon, pangenomes will be built from larger collections of high-quality de novo assembled genomes using accurate long reads. We hope such human pangenomes will enable more comprehensive genotyping of common complex variants (including SVs) from existing catalogs of short-read sequencing data, allowing for the typing of such variants at the scale of existing catalogs of point variation. We expect that unlocking this latent information will ultimately aid with disease association studies and help us further understand how the architecture of the genome contributes to an individual’s phenotype.

## 4.7 Methods summary

### 4.7.1 Evaluation

#### 4.7.1.1 Read simulation

To evaluate Giraffe for mapping human data, we obtained paired-end sequencing reads from a parent-child pedigree. Reads were obtained from an Illumina NovaSeq 6000 machine for parent NA19239 (accession ERR3239454) and from Illumina HiSeq 2500 and HiSeq X Ten machines for child NA19240 (accession nos. ERR309934 and SRR6691663, respec-

tively). These samples were selected because NA19240 has genotypes for the HGSVC variants [15], whereas NA19239 has genotypes for the 1000GP variants [5]. NA19239 was excluded from the 1000GP graph (Section 4.9.2.1). We simulated 1 million read pairs (2 million reads) from each individual’s haplotypes (Section 4.9.2.4).

#### 4.7.1.2 Read mapping accuracy

Simulated read sets were mapped to the graphs using Giraffe, VG-MAP [37], HISAT2 [47], and GraphAligner [72]. We were unable to build a HISAT2 index for the full 1000GP graph, and so instead we mapped it to a graph created from a subset of the 1000GP data where all variants with a frequency below 0.001 were filtered out. In addition, we mapped the read sets to the primary graphs using Giraffe and to the linear reference assemblies using the linear sequence mappers BWA-MEM [53], Bowtie2 [50], and Minimap2 [55]. Mapping accuracy was evaluated by comparing the positions along embedded, shared linear paths at which reads fell after mapping with similarly determined positions for their original simulated alignments.

#### 4.7.1.3 Read mapping speed

We compared mapping runtime, speed, and memory usage on an AWS EC2 i3.8xlarge node with 32 vCPUs and 244 GB of memory. To estimate real-world runtime and memory usage, we aligned a shuffled read set of 600 million NovaSeq 6000 reads from NA19239. We mapped reads to the 1000GP graph, the HGSVC graph, and the GRCh38 linear reference for comparison and measured runtime and memory usage. For each tool, we also separately measured reads mapped per thread per second, ignoring the start-up time of the mapper (Fig.4.23).

This measure gives an estimate of speed that is invariant to read-set size or thread count, except for the effects of long-running work batches and thread synchronization overhead (Section 4.9.2.7).

#### 4.7.1.4 Read-mapping bias

To assess reference allele mapping bias, we mapped 600 million real paired-end NovaSeq 6000 reads for NA19239 to the 1000GP graph using default Giraffe and VG-MAP. For comparison, we mapped the same reads to GRCh38 with BWA-MEM.

#### 4.7.1.5 Genotyping accuracy

We compared the performance of Giraffe, VG-MAP, Illumina’s Dragen platform, and BWA-MEM for genotyping SNVs and short indels. The design of each calling pipeline is described in section 4.9.4 and the parameters and indexes for each experiment are described in Table 4.22. The variants produced by each pipeline were compared against the GIAB v4.2.1 HG002 high-confidence variant-calling benchmark [89] using the RealTimeGenomics vcfeval tool [21] and Illumina’s hap.py tool [4]. This benchmark set covers 92.2% of the GRCh38 sequence.

We also evaluated a DeepVariant [65] pipeline that uses Giraffe mappings (Section 4.9.4.4). Using the default DeepVariant 1.1.0-trained model, we tested genotyping of the HG003 sample across the entire genome. This sample was not used in training the model.

#### 4.7.1.6 Generalization to yeast

To evaluate Giraffe’s performance on more diverged, nonhuman data, we used a yeast graph built from a Cactus multiple sequence alignment for five strains of the *S. cerevisiae* and *S. paradoxus* yeasts [41]. For the corresponding negative-control primary graph, we used the S.c. S288C assembly. We collected basic statistics about the yeast graph and decomposed the graph for analysis using the method of [63]. We simulated 500,000 read pairs from a held-out *S. cerevisiae* yeast strain, DBVPG6044, not included in the yeast graph, using an error and length model for Illumina HiSeq 2500 reads (Section 4.9.2.4).

#### 4.7.1.7 SV genotyping

We built an SV pangenome from the HGSVC [15], GIAB [94], and SVPOP [8] sequence-resolved catalogs. After filtering out erroneous duplicates using a remapping approach, the SVs were iteratively inserted in the genome graph to minimize the effect of errors and redundancy in the catalog. The SVs were then genotyped across 5202 genomes by aligning short-read sequencing data using Giraffe with a workflow description language (WDL) workflow that we deposited in Dockstore [60]. Two-thousand samples were selected from the MESA cohort to maximize sample diversity. The remaining 3202 samples are from the 1000 Genomes Project and include 2504 unrelated individuals. The trios available in this latter dataset were used to compute the rate of Mendelian concordance in the genotypes.

The different SV alleles observed in the population were clustered into SV sites based on their reciprocal overlap (for deletions) and sequence similarity (for insertions). We used the frequency profile across alleles within an SV site to identify the major allele and to fine-

tune variants with near duplicates in the combined catalog that may have been due to errors. Each variant was then annotated with its presence in existing SV databases [8, 84, 38], its repeat content, and its location relative to gene annotations. We also compared the frequency distributions across the SV databases and how well the frequency estimates matched for variants shared across databases.

PCA was performed on the SV genotypes, and principal components were compared with those produced from SNV-indel genotypes. We defined strong intercluster or inter-superpopulation frequency patterns by a frequency in any cluster or superpopulation differing by more than 10% from the median frequency across all of them. For the 2000 MESA samples, the clusters were defined using hierarchical clustering on the first three principal components. For the 1000 Genomes Project, we used their “superpopulation” assignments. Permutations were used to contrast the number of SVs with such patterns with an expected baseline.

Finally, we examined the SV genotypes in a subset of the samples that had gene-expression data available from the GEUVADIS consortium [87]. MatrixEQTL [76] identified SV-eQTLs while controlling for sex and population structures, as summarized by the first four principal components. Separate analyses of the four European-ancestry populations together and the YRI population alone were performed similarly. In addition, we performed a joint eQTL analysis with publicly available SNVs and indels [14]. We used permutation to compute enrichment of SV-eQTLs in gene regions, gene families, or among lead-eQTLs (those with the strongest association for a gene).

## **4.8 Acknowledgments**

We acknowledge the studies and participants who provided biological samples and data for the TOPMed project. The views expressed in this manuscript are those of the authors and do not necessarily represent the views of the National Heart, Lung, and Blood Institute (NHLBI); the National Institutes of Health (NIH); or the US Department of Health and Human Services.

### **4.8.1 Funding**

Research reported in this publication was supported by the NIH under award numbers U41HG010972, R01HG010485, U01HG010961, OT3HL142481, OT2OD026682, U01HL137183, and 2U41HG007234. Research reported in this publication was supported by the NHLBI Bio-Data Catalyst Fellows Program of the NIH through the University of North Carolina at Chapel Hill, under award number OT3HL147154. J.A.S. was supported by the Carlsberg Foundation. Computational resources for the project were made available by the NIH and by Amazon Web Services, without full compensation at market value. The high-coverage sequencing data for the 1000 Genomes Project were generated at the New York Genome Center with funds provided by National Human Genome Research Institute (NHGRI) grant 3UM1HG008901-03S1 and can be found on Terra. MESA and the MESA SHARe projects are conducted and supported by the NHLBI in collaboration with MESA investigators. Support for MESA is provided by contracts 75N92020D00001, HHSN268201500003I, N01-HC-95159, 75N92020D00005, N01-HC-95160, 75N92020D00002, N01-HC-95161, 75N92020D00003, N01-HC-95162, 75N92020D00006,

N01-HC-95163, 75N92020D00004, N01-HC-95164, 75N92020D00007, N01-HC-95165, N01-HC-95166, N01-HC-95167, N01-HC-95168, N01-HC-95169, UL1-TR-000040, UL1-TR-001079, and UL1-TR-001420. Funding for SHARe genotyping was provided by NHLBI contract N02-HL-64278. Genotyping was performed at Affymetrix (Santa Clara, CA, USA) and the Broad Institute of Harvard and MIT (Boston, MA, USA) using the Affymetrix Genome-Wide Human SNP Array 6.0. This work was also supported in part by the National Center for Advancing Translational Sciences, CTSI grant UL1TR001881, and the National Institute of Diabetes and Digestive and Kidney Disease Diabetes Research Center (DRC) grant DK063491 to the Southern California Diabetes Endocrinology Research Center. Whole-genome sequencing (WGS) for the TOPMed program was supported by the NHLBI. WGS for “NHLBI TOPMed: Multi-Ethnic Study of Atherosclerosis (MESA)” (phs001416) was performed at the Broad Institute of MIT and Harvard (3U54HG003067-13S1 and HHSN268201500014C). Core support, including centralized genomic read mapping and genotype calling, along with variant quality metrics and filtering were provided by the TOPMed Informatics Research Center (3R01HL-117626-02S1; contract HHSN268201800002I). Core support, including phenotype harmonization, data management, sample-identity quality control, and general program coordination, was provided by the TOPMed Data Coordinating Center (R01HL-120393; U01HL-120393; contract HHSN268201800001I).

#### **4.8.2 Author contributions**

Project design: D.H., E.G., B.P. Giraffe implementation: J.S., X.C., A.M.N., J.M.E., B.P. SV analysis: J.M., G.H. Short-variant analysis: C.M., P.-C.C., A.C. The vg implementa-

tion: J.S., J.M., X.C., A.M.N., J.M.E., C.M., J.A.S., G.H., D.H., E.G., B.P. Manuscript writing: J.S., J.M., X.C., A.M.N., J.M.E., C.M., J.A.S., G.H., B.P. Data production: N.G., S.G., T.W.B., A.R., K.D.T., S.S.R., J.I.R.

#### 4.8.3 Competing interests

P.-C.C. and A.C. are employees of Google and own Alphabet stock as part of the standard compensation package. The remaining authors declare no competing interests.

subsectionData and materials availability An overview of the data generated for this paper, and key input data to reproduce the analyses, is available at <https://cglgenomics.ucsc.edu/giraffe-data/>. The dataset is available through InterPlanetary File System (IPFS) at <https://ipfs.io/ipfs/QmVo4Q5hCKqUGJJZ>. Archived copies of the code and final reusable work products have been deposited at Zenodo [81]. This archive also includes vg, toil-vg, and toil source code and Docker containers used in this work, as well as the giraffe-sv-paper orchestration scripts. “Final” versions of vg and toil-vg, including all features needed to reproduce this work, are 9907ab2 for vg and 99101f2 for toil-vg. The latest version of the vg toolkit, including the Giraffe mapper, is customarily distributed at <https://github.com/vgteam/vg>. The scripts used for the analysis presented in this study were developed at <https://github.com/vgteam/giraffe-sv-paper>, a git bundle of which is archived at Zenodo [81]. Data used in the Giraffe read-mapping experiments—including the 1000GP, HGSCV, and yeast target graphs, the linear control graphs, the graphs used to simulate reads, and the simulated reads themselves—can be found at <https://cgl.gi.ucsc.edu/data/giraffe/mapping/>. The SV pangenomes and SV catalogs annotated with allele frequencies are hosted at <https://cgl.gi.ucsc.edu/data/giraffe> and archived at Zenodo [81]. This repository also includes SVs with strong inter-superpopulation

frequency patterns, SV-eQTLs, and SVs that overlap protein-coding genes. To build the 1000GP and HGSVC graphs, we used the GRCh38 no-alt analysis set (accession no. GCA\_000001405.15) and the hs38d1 decoy sequences (accession no. GCA\_000786075.2), both available from the National Center for Biotechnology Information (NCBI), in addition to the variant call files distributed by the respective projects. To train read simulation and evaluate speed, we used human read sets ERR3239454, ERR309934, and SRR6691663 and yeast read sets SRR4074256, SRR4074257, SRR4074394, SRR4074384, SRR4074413, SRR4074358, and SRR4074383, all available from Sequence Read Archive (SRA). The public high-coverage sequencing dataset from the 1000 Genomes Project [14] is available at [www.internationalgenome.org/data-portal/data-collection/30x-grch38](http://www.internationalgenome.org/data-portal/data-collection/30x-grch38), including European Nucleotide Archive (ENA) projects PRJEB31736 and PRJEB36890. The gene-expression data were download from ArrayExpress E-GEUV-1 (GD462.GeneQuantRPKM.50FN.samplename.resk10.txt.gz). We downloaded the call sets from the ENCODE portal [82] ([www.encodeproject.org/](http://www.encodeproject.org/)) with the identifier ENCFF590IMH. Individual WGS data for TOPMed whole genomes are available through dbGaP. The dbGaP accession no. for MESA is phs001416. Data in dbGaP can be downloaded by controlled access with an approved application submitted through their website: [www.ncbi.nlm.nih.gov/gap](http://www.ncbi.nlm.nih.gov/gap).

## 4.9 Supplementary materials and methods

### 4.9.1 Giraffe Algorithm

#### 4.9.1.1 Definitions

We work in the *sequence graph* model, as defined in the vg toolkit[37]. In a sequence graph, each node  $v$  is labeled by a nucleotide sequence  $\ell(v)$ . A sequence graph is a bidirected graph, which means that each node  $v$  has two *sides*: left side  $v^-$  and right side  $v^+$ . Edges connect unordered pairs of sides of nodes. Nodes can be *visited* in either forward or reverse *orientation*. When we visit a node  $v$  in forward orientation  $\vec{v}$ , we enter it from the left side  $v^-$ , read the node's sequence  $\ell(\vec{v}) = \ell(v)$ , and exit from the right side  $v^+$ .

For such a visit,  $v^-$  is the *entry side*, and  $v^+$  is the *exit side*. In a reverse orientation visit  $\overleftarrow{v}$ , we enter from  $v^+$ , read the sequence  $\ell(\overleftarrow{v}) = \text{REVCOMP}(\ell(v))$ , the reverse complement of the node's sequence, and exit from  $v^-$ . See Figure 4.8A for an example. A *position* in a sequence graph is a pair  $(x, i)$ , where  $x$  is a node in a specific orientation and  $i$  is an offset in sequence  $\ell(x)$ . We start counting sequence offsets from 0.

We calculate *alignment scores* with the following parameters: +1 for each matching character, -4 for each mismatch, -6 for the first character in an insertion or a deletion, -1 for each additional character in the indel, and +5 at each end if the alignment matches or mismatches at that end of the read. Note that, in this scoring model, indels only occur between matches or mismatches; deletions abutting the ends of the read are always removable to get a higher score (and thus never occur), and insertions abutting the ends of the read are actually *softclips*, and are scored as 0 points.

The discussion below describes the general principles used in the Giraffe mapper. We skip many details and special cases.

#### 4.9.1.2 Graph representation

Our graph representation is based on a bidirectional GBWT index [79]. The GBWT is a data structure, based on the FM-index [36], that can efficiently compress and store a large number of similar paths (haplotypes) in a graph. The primary query in the GBWT is a counting query: as we traverse a path in the graph, we can always tell how many indexed paths contain the path traversed so far as a subpath.

The GBWT data model is based on directed graphs. However, we can accommodate a bidirected graph with a bidirectional GBWT index. For each sequence graph node  $v$ , we have separate GBWT nodes for visits  $\overrightarrow{v}$  and  $\overleftarrow{v}$ . The outgoing edges of a GBWT node correspond to the graph edges on the exit side of that visit, and the incoming edges correspond to the graph edges on the entry side of the visit. See Figure 4.8 B for an example.

A GBWT index encodes the topology of the graph induced by the indexed haplotypes. This is not always identical to the original sequence graph, as some nodes and edges may not be visited by any haplotype. In order to support the *handle graph* interface[32], we need some additional structures:

- Determining whether a node with a given identifier exists in the graph is relatively slow with the GBWT interface. Hence, we cache this information in a bitvector.
- We concatenate the sequences for all GBWT nodes (i.e.  $\ell(\overrightarrow{v})$  and  $\ell(\overleftarrow{v})$ ) for each se-

quence graph node  $v$ ) and store them in a single character array.

- We use an integer array to store references into the concatenation that let us find the beginning of each sequence. When node sequences are requested, we use these references to provide clients with direct access to the stored sequence bytes, without needing to decompress the sequences or create temporary copies.

Accessing edges using the GBWT interface is somewhat slow. We first have to find the record corresponding to the GBWT node and then decompress the header of the record. As seed extension, dynamic programming, and pair rescue repeatedly access the edges in a small subgraph, we can speed them up by caching the decompressed records. We use an application-specific per-thread cache in Giraffe, where eviction consists of discarding the entire cache after each stage of processing for each cluster of aligner seed hits. The cache stores all records we have accessed during the current stage for the current cluster.

#### 4.9.1.3 Downsampling the haplotypes

Mapping to a GBWT with too many haplotypes can be slow in regions with a high density of variations, and can be inaccurate due to rare variants creating false positive mappings. Conversely, there may be regions not covered by the haplotypes in the GBWT, especially unplaced or unlocalized contigs, decoy contigs, and other contigs that might not appear in input VCF files. We therefore provide a mechanism for creating artificial haplotype paths for a graph, with three operating modes:

1. Create an artificial *path cover* of each graph component, assuming that each path is

equally likely to be a true haplotype.

2. *Augment* an existing GBWT with an artificial path cover of graph components with no haplotypes.
3. *Downsample* the haplotypes proportionally in graph components that have them and create an artificial path cover of components with no haplotypes.

We generate the paths with a greedy algorithm similar to that used by the Pan-genome Seed Index[39]. We create  $n$  paths in each weakly connected component and consider the frequencies of subpaths (local haplotypes) of length  $k = 4$  nodes. By default, we use  $n = 64$  in the downsampling mode and  $n = 16$  in other modes. The algorithm generates paths in the directed GBWT graph and later transforms them into paths in the bidirected sequence graph. If the component is acyclic, we start from a source node and extend forward until we reach sink node. Otherwise, we start from an arbitrary node and extend in both directions, stopping when there are no possible extensions in either direction or the length of the path reaches the size of the component.

Assume that we are extending the path forward and we have not reached the stopping condition. Let  $X$  be the  $(k - 1)$ -node suffix of the current path and  $u$  the last node. For any subpath  $P$ , let  $f_s(P)$  be the frequency of  $P$  in the set of paths we have generated so far. We consider all outgoing edges  $(u, v)$  and choose the one with the highest weight  $w(X, v)/(f_s(Xv) + 1)$ , where  $w(\cdot, \cdot)$  is a weight function. If we are downsampling the haplotypes proportionally, we use  $w(X, v) = f_h(Xv)$ , where  $f_h(P)$  is the frequency of subpath  $P$  in the true haplotypes. Otherwise, we use  $w(X, v) = 1$  and try to sample all length- $k$  subpaths as equally as possible.

When we supplement input haplotypes with path covers for components of the graph without them (such as decoy sequences), we call the resulting GBWT a *full* GBWT.

#### 4.9.1.4 Downsampling evaluation

To evaluate the effect of haplotype downsampling on read mapping, we generated GBWT indexes containing different numbers of haplotypes. The *full* 1000GP GBWT index contains the 5,006 input haplotypes (real haplotypes from 2,503 samples, which were restricted to the autosomes and sex chromosomes) and an additional 16 path covers over connected components not containing input haplotypes. We downsampled these input haplotypes to create GBWT indexes containing between 1 and 128 haplotypes, and used the read mapping simulations described to compare these GBWT indexes to the full GBWT index containing all haplotypes and to a primary graph containing only the paths of the linear reference (Figure 4.16A). The mapping benefit of adding more haplotypes was found to plateau at 64 haplotypes, with higher accuracy than is achieved by mapping to the full haplotype set.

In contrast to the 1000GP graph, the full HGSVC GBWT index contains just 6 haplotypes for its three samples. We used this graph for an experiment on generating path covers without known haplotypes, to test how Giraffe, which requires a GBWT, might perform on graphs where there are no haplotypes available. Repeating the same read mapping analysis, we found that path covers alone did not outperform the full underlying haplotype set for the HGSVC graph, but came close to matching its performance (Figure 4.16B).

Similar results were seen for both experiments in paired end mapping mode (Figure 4.16C-D). Consequently, we selected the 64-haplotype GBWT for use in the 1000GP graph

mapping experiments, and the full GBWT for use in the HGSVC graph mapping experiments.

#### 4.9.1.5 Minimizer seeds

Giraffe maps reads using the common seed-and-extend approach. We use a minimizer index[73] for finding seeds. We define a  $(w, k)$ -minimizer as the  $k$ -mer with the smallest hash value in a window of  $w$  consecutive  $k$ -mers and their reverse complements (in a  $k + w - 1$  bp window). As we expect that sequencing errors are rare and that common variants are present in the haplotypes, we use longer than usual minimizers:  $k = 29$  and  $w = 11$  by default.

Our minimizer index is a hash table with 64-bit keys and 128-bit values. 62 bits of the key are used for encoding the  $k$ -mer, allowing us to support  $k \leq 31$  bp seeds. The value is either a single *hit* or a pointer to a sorted array of hits. We use the high bit in the key for indicating the type of the value. For each hit, we use 64 bits for the graph position (53 bits for node identifier, 1 bit for node orientation, and 10 bits for sequence offset) and reserve the other 64 bits for *payload*.

We index only unambiguous minimizers, skipping  $k$ -mers containing characters that represent multiple nucleotides. We only consider paths consistent with the haplotypes. Index construction is fast: typically 5–10 minutes for a human genome graph using 16 threads.

For mapping, we prioritize the most informative minimizers (those with the smallest number of hits) and always select more-informative minimizers before less-informative ones. For a minimizer with  $n$  hits, we define *minimizer score* as  $\max(1 + \ln C_h - \ln n, 1)$ , where  $C_h$  is the *hard hit cap* (default 500 hits). We try to select enough minimizers so that the sum of minimizer scores for the selected minimizers reaches the *score fraction* (default 0.9) of the total

score for all minimizers available, but we never select any minimizers with more hits than the hard hit cap. To do this, we first select all minimizers that have a number of hits at or below the *soft hit cap*  $c_h$  (default 10 hits). Then, if we have not yet achieved the score fraction, we select progressively less-informative minimizers one by one. We stop when we reach the score fraction, run out of minimizers, or the next minimizer has more than  $C_h$  hits.

#### 4.9.1.6 Minimum distance clustering

In order to avoid redundant work, we cluster the seeds into sets that should correspond to the same mapping. Two seeds are assigned to the same cluster if the minimum distance between them is sufficiently small. It must be no larger than the *distance limit*. The distance limit is 200 bp by default or, for longer reads,  $|R| + 50$  bp, where  $|R|$  is the read length.

As computing distances between graph positions can be slow, we use a minimum distance index [16] to speed up the clustering. The index is based on the *snarl decomposition* of the graph. A *snarl*[63] is a graph-aware concept of a genetic locus, or a “site”. It is a subgraph separated by two node sides from the rest of the graph. A *chain* is a series of snarls arranged end to end, like beads on a string, with no connections to anything else except (possibly) at the ends. Any sequence graph can be recursively decomposed into a tree of snarls containing chains, and chains containing snarls. In order to find the distance between two graph positions, we determine their lowest common ancestor in this *snarl tree* and calculate the distance by traversing the tree and consulting precomputed tables at each level.

Non-informative seeds are often scattered around the graph. Finding them in the snarl tree causes cache misses for each seed. However, in a typical sequence graph, most positions

are located close to the root of the snarl tree. For such positions, the amount of information we get from the distance index is small. Hence, we can cache the distance information for seeds near the root in the payload field of the minimizer index.

After clustering, we score the clusters by two criteria and pick the highest-scoring clusters for extension. The primary criterion is *read coverage*: the fraction of the read covered by the seeds. The secondary criterion, for breaking ties, is *cluster score*: the sum of minimizer scores for the minimizers present in the cluster.

#### 4.9.1.7 Gapless haplotype-consistent seed extension

We need to turn seeds into *extensions*: gapless, haplotype-consistent partial alignments that may be longer than a minimizer and may contain some mismatches. We define a *mismatch bound* (default  $d = 4$ ) that limits the number of mismatches in various parts of the extension.

Before extending, we first merge redundant seeds. A set of seeds is redundant if they all correspond to the same gapless alignment between the read and a node.

We then try to extend the remaining seeds into haplotype-consistent alignments without gaps. We extend up to 800 clusters (*extension bound*) but ignore those with read coverage or cluster score a certain threshold below the best cluster. For each cluster, we return a set of extensions. We also determine the *extension set score* for the cluster as an upper bound for the alignment score, assuming that we can chain the extensions arbitrarily over the read while ignoring graph topology.

To find the extensions, for each merged seed, we first gaplessly align the read to

the entire node corresponding to the seed, and place that extension into a priority queue by alignment score. Then we process the extensions in the priority queue, highest-scoring first.

We define a *right extension* of an extension as the original extension, made longer on its right side in a haplotype-consistent way by extending the gapless alignment into another node in the graph. A right extension of an extension matches some nonempty subset of the haplotypes that match the original extension. Each haplotype in the original extension either continues on into some right extension, or ends. An extension is *right-maximal* if i) we have reached the end of the read; ii) it does not have right extensions for all haplotypes that currently match it; or iii) a right extension would have both more than  $d$  mismatches and more than  $d/2$  mismatches in the right flank after the initial node. *Left extensions* and *left-maximality* are defined similarly.

If the highest-scoring extension has not been found right-maximal, we find all its right extensions and place them in the priority queue. If, while attempting this process, we find that the extension is right-maximal, we mark it as such and return it to the priority queue. If the highest-scoring extension *has* been found right-maximal, we extend it to the left as above, marking it left-maximal if we find it to be so. At the end, we report the highest-scoring, both left-maximal and right-maximal extension we find for the seed.

We then look at the collection of highest-scoring, left-and-right-maximal extensions across all seeds. If a full-length alignment with at most  $d$  mismatches exists as an extension of a seed, we will have found it. In that case, we sort all the full-length alignments we have found in descending order by alignment score. Then we greedily select all alignments that do not overlap with any of the already selected alignments, where we consider two alignments *overlapping* if

the fraction of (read, reference) base mapping pairs in common is greater than the *overlap threshold* (default 0.8). If there are no full-length alignments with at most  $d$  mismatches, on the other hand, we trim the partial extensions to maximize the alignment score and return the whole set of distinct extensions.

#### 4.9.1.8 From extension sets to alignments

We transform at least two, but no more than 8 (the *alignment bound*) of the highest-scoring extension sets into alignments, and ignore those with scores further than the *extension set score threshold* (default 20 points) below that of the best set.

We try to find at least two alignments from each extension set we use. If an extension set consists exclusively of full-length gapless alignments, we can use them directly. We take the top two if there are two or more, or the only one otherwise. If there are partial extensions in the set, we also consider alignments computed from the best partial extensions in the set, to see if they beat out any full-length gapless alignments for those top two spots. Using each partial extension as a seed, we perform dynamic programming alignment of partial extensions above a score threshold, which is set dynamically such that we align at least one partial extension and end up with at least two alignments overall (including the full-length gapless ones).

Once these criteria are met, we do not align further partial extensions, unless we estimate that the extension could produce a better alignment than the best we have already seen in the extension set. We estimate the best possible alignment score for extension  $X$  by considering three types of alignments for each unaligned tail, without taking the graph into account:

1. An exact match of at most  $k + w - 2$  bp (shorter than the minimizer window) followed by a gap to  $X$ .
2. A gap to another extension  $Y$ , an exact match up to the first mismatch in  $Y$ , and then another gap to  $X$ .
3. A gap to another extension  $Y$ , then  $Y$  or a part of it, and then another gap to  $X$ . In this case, we assume that all mismatches in  $Y$  are in the part we use.

Then we use the highest score across all cases as the estimate.

To actually perform dynamic programming alignment for an extension, we start from the GBWT search state for the extension. The search state corresponds to a set of haplotypes. For each unaligned tail of the read outside the partial extension, we build a tree-shaped graph by tracing out these haplotypes in the appropriate direction, splitting whenever two or more haplotypes in the set diverge from each other. We then align the tail to the tree using the dozeu library <https://github.com/ocxtal/dozeu>, a vectorized implementation of X-drop alignment.

After collecting up to two alignments from each extension set, we select the highest-scoring alignment as the *primary* for the read, while all others are *secondaries*.

#### 4.9.1.9 Paired-end mapping

Paired-end mapping runs start with fragment length estimation, if a fragment length distribution has not been specified. We map the reads in single-ended mode using a single thread until we have 1000 uniquely mapping read pairs. We then estimate the mean and standard

deviation of fragment length using the uniquely mapping pairs (see Section 4.9.1.10).

Once we have estimated the fragment length distribution, we continue mapping with multiple threads. We cluster the seeds for both reads in the pair at the same time and try to group clusters that are at the right distance from each other. Each *cluster group* can contain clusters from either mate pair that are closer than the *fragment distance limit* to each other (default: fragment length distribution mean + 2 standard deviations). We produce alignments within each cluster as is done for single-ended mapping, prioritizing cluster groups containing clusters from both reads. We then attempt to find pairs of alignments that originated from the same cluster group. Pairs of alignments are scored as the sum of the two alignment scores and the log likelihood of their fragment length. If we do not find paired alignments or if we have an unpaired alignment with score at least 0.9 times (the *rescue threshold*) the score of the best alignment, we try to *rescue* the pair. We attempt to rescue at most 15 unpaired alignments (the *rescue bound*).

To perform rescue, we use an aligned read to find an alignment for its unaligned mate. We use the distance index to find the *rescue subgraph* with all nodes within at most 4 standard deviations of the expected read location. We then find all minimizer hits in the subgraph, including those we ignored during mapping. Because the hits are sorted by node identifier in the index, we can find the ones in the rescue subgraph efficiently by doing list intersection with exponential search. We treat all the seeds as a single cluster and try to extend them without gaps. If we find a full-length alignment, we use it as the rescued alignment.

If there are no full-length extensions, we resort to dynamic programming. We now consider alignments to the entire subgraph, not just to the paths consistent with haplotypes. By

default, we use the dozeu algorithm. If we had minimizer hits in the rescue subgraph, we use the best extension as the seed for dynamic programming. Otherwise, we find the best alignment for the first 15 bp of the read and use it as the seed. As a slower but more accurate alternative to dozeu, we also support using the GSSW alignment implementation[91]. While dozeu must extend a seed, GSSW always finds the best alignment in the subgraph.

#### 4.9.1.10 Fragment length estimation

We use a method of moments approach to estimate the fragment length distribution. The distribution is modeled as a normal distribution. To estimate the parameters of the distribution, we map pairs of reads without pairing constraints until obtaining  $N$  uniquely mapped pairs that can reach each other through some path in the graph (default  $N = 1000$ ). To avoid producing a fragment length distribution that does not reflect expected lengths from paired-end sequencing machines, we do not consider distances that are greater than the *maximum fragment length* (default 2000 bp). We discard the largest and smallest  $(1 - \gamma)/2$  distances that we find (default  $\gamma = 0.95$ ). We use the remaining  $\gamma N$  distance measurements as a sample from a truncated normal distribution. The truncated distribution has the same  $\mu$  and  $\sigma^2$  as the underlying normal distribution, and the truncation points can be calculated based on  $\mu$ ,  $\sigma^2$ , and  $\gamma$ . We can estimate the following parameters using the method of moments:

$$\hat{\mu} = \bar{x} \quad \text{and} \quad \hat{\sigma}^2 = s^2 \left( \frac{1 - 2\alpha\phi(\alpha)}{\gamma} \right)^{-1}$$

In the above,  $\bar{x}$  and  $s^2$  are the maximum likelihood estimates of the mean and variance among the retained  $N$  measurements, and  $\alpha = \Phi^{-1}((1 - \gamma)/2)$  is the left truncation point on a standardized

normal distribution.

#### 4.9.1.11 Mapping quality estimation

Mapping uncertainty is conventionally summarized as a *mapping quality*: the Phred-scaled[34] posterior probability of a mapping error. To compute mapping qualities, we take advantage of the interpretation of an alignment score as the log-likelihood of a Viterbi path through a pair hidden Markov model [28]. With this model, the posterior probability of the maximum likelihood alignment  $\hat{A}$  of a query sequence  $Q$  to the reference  $R$  can be expressed as

$$\begin{aligned} P(\hat{A}|Q,R) &= \frac{P(Q,R|\hat{A})}{\sum_A P(Q,R|A)} \\ &= \frac{\exp \lambda S(\hat{A})}{\sum_A \exp \lambda S(A)}, \end{aligned}$$

where  $S$  is the alignment score, and  $\lambda$  is a normalizing factor that can be numerically computed from the scoring parameters [46]. In practice, we do not score all possible alignments in order to evaluate this expression's denominator. Rather, we approximate it by using the high-scoring secondary alignments that are discovered during the mapping process.

#### 4.9.1.12 Mapping quality caps

We impose several *caps* on the mapping quality derived from the formula above, each of which models a source of error due to Giraffe's heuristics. All mapping qualities are capped at 60 to account for some miscalibration and approximation in the mapping quality model. Another cap is computed based on the probability that all the minimizers we explored were

actually created by base errors. Finally, we employ additional mapping quality caps for paired-end mapping to account for uncertain pairing and error introduced by hard caps in the rescue algorithm.

#### 4.9.1.13 Minimizer error mapping quality cap

The Giraffe algorithm can only find a correct mapping if the read contains instances of minimizers that exactly match minimizers in the true placement in the graph, which then form a cluster, which is then extended to produce an alignment. Consequently, the probability that all the minimizer instances in extended clusters were actually created by errors in the read is an upper bound on the mapping quality. We take a conservative approach to estimating this cap where we actually compute a lower bound on the probability of error. In particular, we make assumptions that restrict the set of errors we consider for the sake of computational efficiency. We consider only substitution errors, which are much more frequent than indel errors, and we restrict our attention to only the most probable way the minimizer instances could have been created, rather all possible ways.

Each minimizer instance in the read is minimal in one or more windows; we call the consecutive run of windows that produce a minimizer instance the minimizer instance's *agglomeration*. The agglomeration can be divided into the minimizer itself, of length  $k$ , and the *flank* of zero or more bases beyond the minimizer on either or both ends. In order for all minimizer instances to be disrupted, it is necessary for each window in each agglomeration to contain at least one error. Mathematically, we say that for an agglomeration  $i$ , the minimizer occurs at bases in the set  $\text{core}(i) = \{\text{minstart}(i) \dots \text{minend}(i)\}$  in the read, and the agglomeration

occurs at bases  $\{\text{aggstart}(i) \dots \text{aggend}(i)\}$ . We write the hash value of the agglomeration's minimizer as  $\text{hash}(i)$ .

We consider only the subset of minimizer instances that were explored, and thus contributed to the set of candidate alignments. Our goal is to find a lower bound on probability of disrupting all minimizers, via errors in their agglomerations. We take advantage of the fact that we are looking for a lower bound by restricting our considered cases to those where each base error disrupts all windows that it overlaps. We also employ an approximation, by treating as sufficient the disruption of any windows of an agglomeration with an error in the flank, instead of requiring errors in the flank to disrupt all windows of the agglomeration.

We consider the minimizers' agglomerations to be sorted in the order they appear along the read. We approach this as a dynamic programming problem, with the dynamic programming table  $c$ , where  $c_{i+1}$  holds a lower bound on the probability that all agglomerations  $\{0 \dots i\}$  were disrupted. We use  $I$  to represent the number of the final agglomeration, and we number the bases in the read (or *columns*) as  $j$  running  $\{0 \dots J\}$ .

We imagine all the agglomerations arranged under the read, each on its own line (Figure 4.9B). We overlay the minimizers with *regions* left to right, where the region changes whenever an agglomeration begins or ends (Figure 4.9C). We number these regions as  $r$  running  $\{0 \dots R\}$ , and define their 0-based end-inclusive bounds in agglomerations as  $\{\text{top}(r) \dots \text{bottom}(r)\}$  and in columns as  $\{\text{left}(r) \dots \text{right}(r)\}$ .

We define the following events to structure our probability model:

**BREAKREGION**( $r$ ) = all agglomerations in region  $r$  are disrupted by errors in the region

**BREAKREGIONBYCOLUMN**( $r, j$ ) = all agglomerations in region  $r$  are disrupted by an error at column  $j$

**ERRORBREAKSAGG**( $i, j$ ) = agglomeration  $i$  would be disrupted if there were an error at column  $j$

**BASEERROR**( $j$ ) = there is an error at column  $j$

**BEATHASH**( $h$ ) = a sequence has a hash more minimal than the hash  $h$

**ANYBEATHASH**( $n, h$ ) = any of  $n$  sequences has a hash more minimal than the hash  $h$

We implement the following recurrence relation on the dynamic programming table  $c$ :

$$c_0 = 1$$

$$c_{i+1} = \max_{\forall r | i \in \{\text{top}(r) \dots \text{bottom}(r)\}} c_{\text{top}(r)} \cdot P(\text{BREAKREGION}(r))$$

In addition to the minimizer and region information, we take the read quality string  $q$  as input, which defines  $P(\text{BASEERROR}(j))$ . Also, we approximate the output of  $\text{hash}(i)$  as a uniform random integer in  $\{0, 1, \dots, 2^{64} - 1\}$ . We can then evaluate the recurrence to fill the dynamic programming table using the following relationships:

$$\begin{aligned} P(\text{BREAKREGION}(r)) &= 1 - \prod_{j=\text{left}(r)}^{\text{right}(r)} (1 - P(\text{BREAKREGIONBYCOLUMN}(r, j))) \\ P(\text{BREAKREGIONBYCOLUMN}(r, j)) &= \left( \prod_{i=\text{top}(r)}^{\text{bottom}(r)} P(\text{ERRORBREAKSAGG}(i, j)) \right) \cdot P(\text{BASEERROR}(j)) \\ P(\text{BASEERROR}(j)) &= 10^{-q_j/10} \\ P(\text{ERRORBREAKSAGG}(i, j)) &= \begin{cases} 1, & \text{if } j \in \text{core}(i) \\ P(\text{ANYBEATSHASH}(\text{candidates}(i, j), \text{hash}(i))), & \text{otherwise} \end{cases} \\ \text{candidates}(i, j) &= \min(j - \text{aggstart}(i) + 1, \text{aggend}(i) - j + 1, k) \\ P(\text{ANYBEATSHASH}(n, h)) &= 1 - (1 - P(\text{BEATHASH}(h)))^{n-1} \\ P(\text{BEATHASH}(h)) &= \frac{h}{2^{64}} \end{aligned}$$

The final result is the bottom cell in the dynamic programming table, at  $c_{I+1}$ : the probability of the highest-probability case in which all agglomerations in the region are dis-

rupted. This probability is converted from the internal log10-probability representation used for computation into a Phred score.

The Phred score is then transformed by an empirically-developed heuristic to produce a mapping quality cap, based on how the computation of the initial mapping quality went for the read or read pair. The initial mapping quality can be *pegged* at the maximum value representable by a 32-bit signed integer, which occurs when the score gap between the top-scoring alignment or pair and any secondary is too great. Our heuristic is that, when the initial mapping quality is not pegged, the cap is used as-is, but if the initial mapping quality for the read is pegged to its maximum representable value, the cap is doubled.

#### 4.9.1.14 Paired read mapping quality caps

We apply two different caps for paired reads based on the outcome of the pair rescue routine. If we didn't find the best pair of alignments with rescue, we cap the mapping quality of each individual alignment to account for uncertain pairing. First, we find all alignments for the one read that were in a pair with the same alignment of the other read. We then calculate the mapping quality of the best pair based on the scores of these alignments and use the resulting value as a cap on the mapping quality of the other read. As a second cap, we treat all equally-scoring cluster groups as having an equal probability of producing a correct pair of alignments. Therefore, for the final alignment pair, we take the probability that we picked the incorrect alignment pair to be at most  $1 - \frac{1}{C}$ , where  $C$  is the number of equivalent or better cluster groups, and cap the mapping quality accordingly.

If all pairs of alignments were found by rescue, then we adjust the mapping quality

to model the effects of Giraffe’s hard limits on the number of rescue attempts. For the primary alignment pair, we estimate the number of equivalent pairs that would have been found without the limit. This estimate is the total number of unpaired alignments divided by the number of alignments that were actually rescued. When calculating the mapping quality of the primary alignment pair, we compute it as if the estimated number of pairs had been found.

### 4.9.2 Mapping evaluation

We ran each of the mappers on a variety of different graphs and read sets to evaluate their accuracy (Figure 4.3 and Figures 4.16, 4.10, 4.11, and 4.12), speed (Figure 4.23), and time and memory consumption (Figure 4.4).

#### 4.9.2.1 Graph construction

Human graphs were constructed using the `toil-vg` tool, versions `71f484c` through `cfe1599`, as well as version `b319a1b`. Construction and indexing for the HGSVC graph required multiple retries of the workflow with successive improved versions of the tool. Some indexes, such as the multiple kinds of sampled GBWT indexes and their corresponding minimizer indexes, were generated with additional ad-hoc scripts around functionality in vg. To run GraphAligner, we also needed to convert our vg-format graphs to GFA format using vg.

Reported variant counts were derived from the VCFs used to build the graphs, with `bcftools stats`. A local graph visualization was produced with [11], for path 11 (GRCh38 chr11) from 105,027,313 bp to 105,027,345 bp, by cutting out the local region in advance with `vg chunk`.

The indexes used by HISAT2 for the mapping evaluation were constructed using version 2.2.0. Variants were first split into bi-allelic sites and normalized using bcftools version 1.9 [52]. The variants were then preprocessed using

`hisat2_extract_snps_haplotypes_VCF.py` (part of the HISAT2 package) with the option that allows variant ids not beginning with rs. Furthermore, Y bases in the reference genome were changed to N for the preprocessing only. The resulting variant and haplotype list was then used as input to `hisat2-build` to construct the index.

For the 1000GP graph, we used VCF files from the 20170504 GRCh38 liftover of the 1000 Genomes Project variants, previously available at [http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/GRCh38\\_positions/](http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/GRCh38_positions/) and currently obtainable from [http://ftp.ensembl.org/pub/data\\_files/homo\\_sapiens/GRCh38/variation\\_genotype/](http://ftp.ensembl.org/pub/data_files/homo_sapiens/GRCh38/variation_genotype/). These VCFs have since been withdrawn in favor of newer call sets produced directly on GRCh38. We used the liftover files anyway, as we had already validated that our graph construction pipeline could work with them, and we were unable to find a new call set that covered all the chromosomes and had the level of inter-sample variant spelling consistency of the lifted-over co-called variants. Additionally, we pre-processed the VCFs to remove variants in segmental duplications over 10 kilobases, as identified in [49] and described by [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/genome-stratifications/v2.0/GRCh38/SegmentalDuplications/GRCh38\\_segdups\\_gt10kb.bed.gz](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/genome-stratifications/v2.0/GRCh38/SegmentalDuplications/GRCh38_segdups_gt10kb.bed.gz). We did this because, when we did not do this, and used the resulting graphs for genotyping, we found evidence of miscalls in these regions; we suspect that these regions may have been particularly difficult to lift over, resulting in misplaced variants that do more harm than good at the variant calling stage. Be-

cause we used reads simulated from NA19239 to evaluate mapping accuracy in the 1000GP graph, we did not include any variants that were unique to NA19239 in the 1000GP graph (their child NA19240 was not present in the call set). NA19239 and associated private variants were filtered out of the input VCF files before the graph was constructed. The other parent of the family, NA19238, was retained. This left 2,503 samples to be included the base haplotype index. Additionally, we produced a sample graph containing only variants and haplotypes from sample NA19239, in versions with and without reference paths through sites where the sample had two alternate alleles, to use as a positive mapping control. We used the graph including the reference path as our positive control.

For the HGSVC graph, VCF files from the HGSVC were used (see Hickey et al.[41] and available [here](#)), together with a FASTA reference combining the GRCh38 no-alt analysis set (accession GCA\_000001405.15) and the hs38d1 decoy sequences (accession GCA\_000786075.2). Although NA19240 was used as the simulation target for simulating reads, no haplotype data or private variants were held out, as the graph contained only three samples to begin with. For a positive mapping control, we used the full graph, but a haplotype database containing only NA19240.

The same reference genomes and variant sets as above were used to build the HISAT2 indexes. However, for the GRCh38 1000GP set we filtered all variants with a frequency below 0.001 since HISAT2 was not able to construct an index using the full variant set.

To allow more direct comparison of Giraffe to linear mappers, we created *primary graphs* containing only the primary linear reference genome (GRCh38) with no variants or alternative loci, which we used as a negative control.

For the yeast experiments, a five-strain yeast multiple sequence alignment previously constructed[41] was imported as a graph with `hal2vg` v2.1[41], with ancestral genomes (internal nodes in the alignment tree) excluded.

#### 4.9.2.2 Sample selection

For human evaluation, we decided to use real and simulated reads from NA19239 and NA19240. These two people are parent and child from a well-studied trio pedigree, sometimes described as “the Yoruba trio”, recruited from among people claiming four Yoruba grandparents in Ibadan, Nigeria[44, 6]. After obtaining informed consent and collecting blood, the original collecting investigators gave donors “an equivalent of ~US \$8.00 and multivitamins worth ~US \$4.00 to compensate them for their time and travel”[43]. Because NA19239 had haplotypes available on the 1000 Genomes variants, and NA19240 had haplotypes available on the HGSVC variants (no sample has both), and real reads from a variety of Illumina instruments were publicly available for one or the other, these two samples were chosen to avoid unduly confounding graph choice and sample genome.

#### 4.9.2.3 Real reads

For speed evaluation on yeast data, real reads for each strain were used, from accessions SRR4074256, SRR4074257, SRR4074394, SRR4074384, SRR4074413, SRR4074358, and SRR4074383, shuffled and subset to 500 thousand pairs as with the human data.

#### 4.9.2.4 Read simulation

To train vg's read simulator for generating human reads, the truncated 500 thousand pair real read sets were used to train the simulator to match the read length and quality string characteristics of each of the three different Illumina instruments. Training reads for human were interpreted as pairs. To train the read simulator for the yeast experiments, the Illumina HiSeq 2500 reads from accession SRR4074257 were used as the training reads. These training reads were not interpreted as pairs, and the entire file was used.

For the human 1000GP experiments, using `toil-vg` construct, the input VCFs were subset to just sample NA19239, and a pair of single-haplotype graphs were built. These graphs were used as input to `toil-vg sim`, trained as above for the different sequencer models, and configured for an average fragment length of 570 bp, a standard deviation of 165 bp, and an indel rate of 0.029%. Reads were annotated with path positions for touched reference paths. Reads touching the hs38d1 general decoy sequence and the NC\_007605 viral decoy were excluded. Finally, we truncated to exactly 1 million pairs (2 million reads).

For the human HGSVC graph, we needed to annotate reads not only with positions along the reference paths, but also with positions along alt allele paths corresponding to large insertions, in order to assess our ability to correctly place reads that did not touch the primary reference. Since the way these paths are named by vg differs depending on the set of alt alleles in the input VCF, the single-haplotype graph approach could not be used. Instead, we simulated from the full HGSVC graph using `vg sim`, restricting the simulator to the haplotype segments representing sample NA19240 in the GBWT. The same training reads, fragment

length distribution, and error parameters were used as in the 1000GP case. Reads were annotated with path positions for all touched paths, including reference and alt allele paths. Because NA19240 is known to have an XX karyotype, we excluded reads annotated as touching paths in the connected component of the graph hosting the Y chromosome reference path. Reads touching paths containing “JTFH” or “KN70” in their names were also excluded, in order to remove reads simulated from decoy sequences. Finally, we truncated to exactly 1 million pairs (2 million reads).

For yeast, reads were simulated from the DBVPG6044 strain as contained in an *all-strain yeast graph*. This graph was generated using `hal2vg v2.1` from a previously reported Cactus alignment of the five strains in the yeast graph and an additional seven strains[41]. The `vg sim` tool was used on this graph standalone, without `toil-vg`. The simulator was trained as described above, and used the same fragment length distribution and indel rate as for the human reads. Simulation was restricted to the paths in the graph belonging to DBVPG6044. Exactly 1 million reads (500,000 pairs) were simulated for the DBVPG6044 strain; no truncation was performed.

#### 4.9.2.5 Read mapping

For our read mapping experiments on human data, we used `vg` release 1.27.1 to run Giraffe and VG-MAP. For mapping to yeast, we used commit 1d9385e. We tested this commit on a small number of cases with human data and saw no difference between it and the 1.27.1 version. For our simulated read mapping experiment on the 1000GP graph (Figure 4.3 A,D, we used a 64 haplotype sampled GBWT and for the HGSVC graph (Figure 4.3 B,E, we used

the full GBWT for mapping with Giraffe. We ran Bowtie2 version 2.4.1 with default settings for single-ended reads, and a maximum fragment length of 1065 for paired-end. BWA-MEM version 0.7.17-r1188 was run with default settings. GraphAligner version 1.0.11 was run with its variation graph parameter preset. HISAT2 version 2.2.1 was run with default parameters with maximum fragment length 1065 and no spliced alignment. We also ran HISAT2 with its sensitive and very sensitive settings. Minimap2 version 2.14-r883 was run with its short reads preset and no secondary alignments. A Docker container with these versions of each of the mappers is available on Docker Hub: [xhchang/vg:giraffe-paper](#)

As a positive control, we mapped reads to our positive control sample graphs using Giraffe. Giraffe was run in a permissive parameterization (`--hit-cap 100 --hard-hit-cap 5000 --score-fraction 1.0 --max-extensions 10000 --max-alignments 1000 --cluster-score 0 --cluster-coverage 0 --extension-score 0 --extension-set 0`), to attempt to exhaustively explore all possible alignments.

#### 4.9.2.6 Evaluating simulated read mapping

We used the path position annotations that were found during simulation to determine the correctness of read mappings. After mapping, each read was annotated again with the nearest reference position in the graph. For regions in the held out sample not included in the reference assemblies, other linear paths were also considered (such as the alt allele paths in the HGSVC graph). Any alignment that was within 100 bp of the true reference position was considered correct. For the linear mappers, we *injected* the alignments into the graph with `vg inject`, turning them into alignments to the graph, and then performed the same evaluation

as for the graph mappers. Several tools allowed split read mapping and produced both primary and supplementary alignments for a single read. In these cases, we counted a read as correctly mapped if any of its alignments were correct.

#### 4.9.2.7 Speed, runtime, and memory evaluation

For our speed, runtime, and memory evaluations we used an AWS EC2 i3.8xlarge node with 32 vCPUs and 244 GB of memory. We found the total runtime and memory use of each of the mappers using our 600-million-read NovSeq 6000 set. Runtime and memory were measured using `/usr/bin/time`. Each of the mappers was run using 16 threads. We also separately measured reads mapped per thread per second, ignoring the start-up time of the mapper (Figure 4.23), but although this metric can be more useful for provisioning resources, it relies on more assumptions and could not be obtained for some mappers. For speed evaluation we mapped reads to the 1000GP graph and GRCh38 linear reference, and to the HGSVC graph and GRCh38 linear reference. We were unable to map this read set with GraphAligner on the 1000GP graph as we ran out of memory.

We found the speed of each mapper using our 1 million read real read sets for each sequencing technology. Each mapper except for Minimap2 was run on 16 threads; Minimap2 was run on 2 threads because it did not use all 16 threads when given. Speed was measured as reads per second per thread based on the time each tool reported as spending on mapping. We did not find the speed of GraphAligner because it logs mapping time per read, and dumping and aggregating the measurements without affecting the mapping speed would not have been feasible.

### 4.9.3 Evaluation of reference allele bias

We produced an allele balance plot (Figure 4.5A) to assess the mapping bias of Giraffe, VG-MAP, and BWA-MEM. We mapped 600 million real paired-end NovaSeq 6000 reads for NA19239 to our 1000GP graph or GRCh38 using each of the mappers. For the graph mappers, Giraffe and VG-MAP, we projected the graph alignments to alignments to the linear GRCh38 reference using `vg surject`. Replicating a previous approach[23], we used `bcftools mpileup` and `bcftools call`[52] to call variants, and filtered to high-confidence variants (root mean square read mapping quality  $\zeta = 40$  and depth  $\zeta = 25$ ) that were called as heterozygous for all mappers. This strategy was chosen to restrict analysis to sites that we are confident are heterozygous in NA19239 and are supported by the read set used. `mpileup` and `call` were run on each chromosome in parallel using GNU Parallel [85].

### 4.9.4 Genotyping and evaluating short variants with Dragen

We mapped  $\sim$ 830 million paired-end, 150-bp-long reads (Precision FDA Challenge V2 Illumina  $\sim$ 35x coverage) from the HG002 sample to the 1000GP graph. We also separately evaluated mapping  $\sim$ 1.08 billion 250-bp-long reads ( $\sim$ 40-50x coverage of Illumina HiSeq 2500) from HG002 to see if using longer reads and higher coverage would affect the results. For the genome graph mappers, we evaluated variant calling performance by using `vg surject` to produce BAM representations of our graph alignments projected onto the linear reference assembly, and then using Dragen version 3.7.5 to call variants against the hs38d1 reference for each set of alignments. Dragen was used as the primary variant caller because Illumina, who sells it, has found it to produce robust results[42, 1, 2]. Its speed was also useful

for the purposes of rapid evaluation of whole genome alignments of real read data. No training or calibration was performed for any of the generated mappings other than those performed by default by Dragen itself.

All pipelines evaluated for short variant calling performance had the same structure. First, we mapped reads to the appropriate GRCh38-based linear or graph reference using the pipeline’s mapper. Then, we genotyped the resulting alignments using Illumina’s “Dragen Bio-IT Platform” product, version 3.7.5, against an index generated from the hs38d1 human genome reference. The mappers for each pipeline evaluated were Illumina’s Dragen internal aligner, BWA-MEM, VG-MAP, Giraffe, and Giraffe in fast mode. The Genome In a Bottle HG002 version 4.2.1 high-confidence VCFs were used as the truth sets for evaluating performance of variant calling[19]. The VCF was obtained from [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002\\_NA24385\\_son/NISTv4.2.1/GRCh38/HG002\\_GRCh38\\_1\\_22\\_v4.2.1\\_benchmark.vcf.gz](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002_NA24385_son/NISTv4.2.1/GRCh38/HG002_GRCh38_1_22_v4.2.1_benchmark.vcf.gz) and the high confidence regions evaluated were based on the BED files obtained from [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002\\_NA24385\\_son/NISTv4.2.1/GRCh38/HG002\\_GRCh38\\_1\\_22\\_v4.2.1\\_benchmark\\_noinconsistent.bed](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002_NA24385_son/NISTv4.2.1/GRCh38/HG002_GRCh38_1_22_v4.2.1_benchmark_noinconsistent.bed). We used RTG’s vcfeval [21] and Illuminas hap.py <https://github.com/Illumina/hap.py> to evaluate variant calling concordance with truth sets. All pipelines used the computational resources of the NIH HPC Biowulf cluster (<http://hpc.nih.gov>).

#### 4.9.4.1 Graph construction

For the variant calling experiments, graph references, and their indexes for running vg mappers, were constructed using different versions of vg ranging from 1.20.0 to 1.31.0, orchestrated by toil-vg construct.

The graph references for GRCh38-based experiments were constructed using vg version v1.31.0 as packaged in container quay.io/vgteam/vg:v1.31.0, using the canonical hs38d1 FASTA reference file as sourced from [https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe\\_manuscript\\_data/genome\\_references/linear\\_references/GCA\\_000001405.15\\_GRCh38\\_no\\_alt\\_analysis\\_set\\_plus\\_GCA\\_000786075.2\\_hs38d1\\_genomic.fna](https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe_manuscript_data/genome_references/linear_references/GCA_000001405.15_GRCh38_no_alt_analysis_set_plus_GCA_000786075.2_hs38d1_genomic.fna) and the 1000 Genomes Project Phase 3 phased VCFs that were lifted over from old GRCh37-based joint-called datasets as sourced from [https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe\\_manuscript\\_data/genome\\_references/graph\\_references/1000gp\\_data/ALL.chr\\*\\_GRCh38.genotypes.20170504.rename.vcf.gz](https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe_manuscript_data/genome_references/graph_references/1000gp_data/ALL.chr*_GRCh38.genotypes.20170504.rename.vcf.gz). Primary graphs and indexes were constructed using just the linear reference FASTA with vg version v1.26.0-180-gdc119fa04 as packaged in container quay.io/vgteam/vg:ci-2284-dc119fa046aa7131a1a8e026be36da2d79bc2f22. 1000GP graphs and indexes were constructing using both the FASTA and the VCFs after filtering out variants belonging to segmental duplication regions of greater than 10 kilobases in length as defined in the bed file [https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/genome-stratification/v2.0/GRCh38/SegmentalDuplications/GRCh38\\_segdups\\_gt10kb.bed.gz](https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/genome-stratification/v2.0/GRCh38/SegmentalDuplications/GRCh38_segdups_gt10kb.bed.gz) with vg version v1.31.0 as packaged in container quay.io/vgteam/vg:ci-2890-655a9622c3d60e87f14b88d943fdb8554214a9

#### **4.9.4.2 Read sets**

Two different read sets were used to evaluate performance. One was the 150 bp paired-end FASTQ set from sample HG002 as obtained from the FDA Precision Challenge dataset. These reads are available as paired FASTQs at [https://storage.googleapis.com/cmarkell-vg-wdl-dev/test\\_input\\_reads/HG002.NovaSeq.pcr-free.35x.R1.fastq.gz](https://storage.googleapis.com/cmarkell-vg-wdl-dev/test_input_reads/HG002.NovaSeq.pcr-free.35x.R1.fastq.gz) and [https://storage.googleapis.com/cmarkell-vg-wdl-dev/test\\_input\\_reads/HG002.NovaSeq.pcr-free.35x.R2.fastq.gz](https://storage.googleapis.com/cmarkell-vg-wdl-dev/test_input_reads/HG002.NovaSeq.pcr-free.35x.R2.fastq.gz). The other read set consisted of 250 bp paired-end FASTQ reads from sample HG002, obtained from the GIAB NovoAligned BAM at [ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002\\_NA24385\\_son/NIST\\_Illumina\\_2x250-bps/novoalign\\_bams/HG002.hs37d5.2x250.bam](ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/NIST_Illumina_2x250-bps/novoalign_bams/HG002.hs37d5.2x250.bam). These reads in paired FASTQ format are available at [https://storage.googleapis.com/cmarkell-vg-wdl-dev/test\\_input\\_reads/HG002\\_read\\_pair\\_1.fq.gz](https://storage.googleapis.com/cmarkell-vg-wdl-dev/test_input_reads/HG002_read_pair_1.fq.gz) and [https://storage.googleapis.com/cmarkell-vg-wdl-dev/test\\_input\\_reads/HG002\\_read\\_pair\\_2.fq.gz](https://storage.googleapis.com/cmarkell-vg-wdl-dev/test_input_reads/HG002_read_pair_2.fq.gz).

#### **4.9.4.3 Read mapping**

The HISAT2 mapping runs were done using version 2.1.0. Only default settings, with the addition of the --no-spliced-alignment flag, were used during HISAT2 execution.

One pipeline used Illumina's Dragen module for both mapping and alignment; both steps were run against the same index derived from the hs38d1 human genome reference.

The BWA-MEM mapping runs were done using version 0.7.17-r1188 against the hs38d1 human genome reference.

The vg alignment runs used different graph indexes depending on the mapper used

(VG-MAP or Giraffe). GRCh38-based experiment run details are given in Supplementary Table 4.22. All runs used the `toil-vg map` workflow for the alignment procedure and ABRA2[61] for indel realignment after alignment. GATK’s Picard tool [3] was used to reorder aligned BAM files for contig consistency which is required by the Dragen variant calling algorithm.

#### 4.9.4.4 DeepVariant calling

All pipelines for the DeepVariant experiment evaluation used BWA-MEM version 0.7.17-r1188 against the hs38d1 human genome reference or VG Giraffe against the same 1000GP graph as described in 4.9.4.1. Alignments were then indel realigned using ABRA2[61]. The DeepVariant v1.1.0 code base was used for all experiments.

DeepVariant learns to call variants by training on input data from gold standard samples HG001-HG007. In all DeepVariant models, the HG003 sample is fully withheld from training, allowing a full genome sample to be evaluated. Because the DeepVariant production models for Illumina are trained on samples mapped with BWA, DeepVariant likely optimizes for the mapping accuracy and quality profile of BWA. We used DeepVariant v1.1.0 production model for the BWA-MEM experiments and a custom model for the GIRAFFE alignments that were trained using the same methods that were used to train the DeepVariant model for BWA-MEM alignments. The custom model data can be found here [https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe\\_manuscript\\_data/deepvariant\\_models/model.ckpt-25600\\*](https://storage.googleapis.com/cmarkell-vg-wdl-dev/giraffe_manuscript_data/deepvariant_models/model.ckpt-25600*).

#### 4.9.5 Genotyping and evaluating structural variants

We replicated the evaluation performed in Hickey et al.[41] using two SV graphs made from the HGSVC[15] and GIAB v0.6[94] SV catalogs, respectively. The graphs had originally been built from VCF files provided by the HGSVC and GIAB consortia. To evaluate Giraffe, we built the necessary indexes. For the minimizer index, we used the default minimizer length of 29 and window size of 11. The GBWT index was created using the greedy path-cover algorithm and  $n = 32$  paths. We then mapped and genotyped the SVs in four samples: HG00514, HG00733 and NA19240 for the HGSVC graph; HG002 for the GIAB graph. The genotyped were compared to the truth-set using `sveval`[41]. We use non-repeat regions, i.e. regions of the genome that don't overlap simple-repeats and segmental duplications, as high-confidence regions for the HGSVC graph and the combined SV graphs that are both on GRCh38. For the GIAB graph, which uses GRCh37, we use the high-confidence regions provided by the GIAB consortium[94]. The scripts to build the pangenomes, genotype the SVs and evaluate the genotyping performance are available in the repository (see Section sec:code-data). (see section 4.8.3).

#### 4.9.6 Combining long-read catalogs for an SV pangenome

We combined variants from the three SV catalogs used previously in Hickey et al.[41]: HGSVC [15], GIAB[94], and SVPOP [8]. The three catalogs were first normalized and merging using bcftools version 1.9 [52].

To minimize issues with near-duplicate variants, we first mapped short reads from 17 diverse samples to a local graph containing all versions of a variant, and kept one version

if substantially supported by the reads. Near-duplicate variants often exists because they contain errors around their breakpoints. Our aim was to filter out the most obvious errors before starting to build the SV graph. We identified near-duplicates by overlapping the SVs in this combined catalog. At this stage, we defined near-duplicate SVs as deletions with a reciprocal overlap at least 80% or insertions located at less than 50 bp from each other and sizes at least 80% reciprocally similar. A network was build from this “near-duplicate” relationships to identify connected components of near-duplicate variants, which we referred to as SV clusters. We identified 34,010 such clusters of near-duplicates SVs, each with on average 2.6 SVs. For each SV cluster, we extracted the reference sequence and variants to create a local SV graph of the region. We then aligned short-reads pooled from 17 diverse individuals selected to span different populations in the 1000 Genomes Project and HGSCV. The reads aligning to the region of interest were extracted using SAMtools[52] from the aligned reads publicly available. A subset of SV clusters didn’t have SV-supporting reads, because they originally couldn’t be mapped to the linear reference, were too difficult to map in general, or the variants were rare enough that none of the 17 individuals carried them. Out of the 23,929 clusters with read support, 43.1% had one variant supported by more than 50% of the SV-supporting reads. When reads supported one variant better than the other near-duplicates in the SV cluster, the near-duplicates were discarded. This filtering step removed 55,330 near-duplicate SVs.

Although many obvious duplicates had been removed, many SVs sharing sequence remained. For example if SVs were not covered by the short-reads used above or if they were too different to match the near-duplicate definition used above. Instead of building a pangenome naively from the variant definition, we used a hybrid approach using iterative graph augmen-

tation. The goal is to collapse as much as possible variants that share sequence content by iteratively aligning and adding variants into the SV graph. The graph was built in chunks separated by at least 10 kbp without any variants. We focused on SVs shorter than 1 kbp which represented the vast majority of the near-duplicate variants and allowed us to parallelize this iterative graph augmentation approach. For each chunk, we clustered SVs using a less stringent criterion as above: deletions with at least 30% reciprocal overlap; insertion located at less than 100bp from each other and with at least 30% reciprocally similar sizes. The 18,286 clusters of SVs identified contained an average of 3.24 SVs. A graph was built for the chunk using one variant per SV cluster. One by one, the other variants were then added by aligning them to the graph and augmenting the graph. We used `vg mpmap` [78] to align large sequences representing the SV variation flanked by 5 kbp of sequence context. The alignment was added to the graph using `vg augment`. The chunked graphs was merged using `vg concat`.

Finally, we added unplaced and unlocated contigs from the GRCh38 reference assembly to the pangenome. We also added the hs38d1 decoy sequences (accession GCA\_000786075.2).

The scripts used to build this pangenomes are documented in the repository (see Section 4.8.3).

#### 4.9.7 Population scale analysis of SVs

##### 4.9.7.1 Whole-genome sequencing datasets

The MESA whole-genome sequencing dataset was obtained from NHLBI's TOPMed program ([www.nhlbiwg.org](http://www.nhlbiwg.org)), freeze 5b. Paired-end 150bp reads had been sequenced on Illumina HiSeq X Ten instruments and achieved a mean depth of at least 30x. All sequencing had

used PCR-free library preparation kits.

For the 1000 Genomes Project samples, we used public whole-genome sequencing data generated at the New York Genome Center[14] (see Section 4.8.3). Samples had been sequenced to a minimum of 30x mean genome coverage using PCR-free sequencing libraries on the Illumina NovaSeq 6000 sequencing instrument, with 150bp paired-end reads. All cell lines had been obtained from the Coriell Institute for Medical Research and had been consented for full public release of genomic data.

Of note, the reads were downsampled to 20x in both datasets and mapped with Giraffe fast mode to reduce the computing cost.

#### 4.9.7.2 Pangenome genotyping workflow

A genotyping pipeline was implemented in WDL and ran using the BioData Catalyst ecosystem[62] providing access to the TOPMed and 1000 Genomes Project datasets. The WDL pipeline was deposited on Dockstore[60] and ran on Terra. TOPMed data was accessed through Gen3. The estimated cost per sample was estimated from Google Billing Reports. Resources used for each sample or step in the workflow were extracted using `terra-utils` notebooks from the [BioData Catalyst Powered by Terra](#). On average, genotyping a sample took 194.4 cpu-hours of compute and cost between \$1.11 and \$1.56 (Figure 4.24A, Tables 4.17 and 4.21). Around 71% of the time is spent mapping reads, 18% genotyping the variants, and the remainder downloading and preparing the raw reads (Table 4.21).

#### **4.9.7.3 Diverse sample selection from the MESA cohort**

The MESA cohort is a medical research study involving white, African American, Hispanic/Latino, and Chinese adults from 6 U.S. communities [12]. We selected a subset of 2 thousand individuals in the MESA cohort from the TOPMed freeze 5b. The individuals were selected to cover as uniformly as possible the space defined by the top 11 principal components provided by the TOPMed consortium. These principal components (PCs) were derived from SNVs genotypes across the full TOPMed dataset. Briefly, the selection process started with the two individuals that located the furthest from each other in the PC space. New individuals were then selected, one by one, to maximize their distance to already selected individuals.

#### **4.9.7.4 Combining SV genotypes across samples**

SVs were merged using bcftools version 1.9 [52]. For each sample, we split multi-allelic variants, normalize them and sort the output VCF. The output of `vg call` is derived from paths in the graph (like haplotypes) that traverse the SV site. For large SV sites, it means that one called variant may contain multiple smaller canonical SVs. In addition, this path-based reporting of the variants can sometimes separate homozygous canonical variants into two heterozygous variants embedded within larger paths. Hence, during normalization, we aligned the reference and alternate alleles in the VCF file to break down each variant into canonical SVs. We also combined heterozygous variants that matched exactly into homozygous calls. These SVs were then left-aligned and sorted using bcftools. Finally, the VCF files were combined using `bcftools merge` using the `-0 -m none` parameters to avoid multi-allelic records. The corresponding WDL workflow has been deposited on Dockstore[59].

#### **4.9.7.5 Clustering SV alleles into SV sites**

Deletions were matched if their reciprocal overlap was at least 80%. Insertions were matched if located at less than 20 and their sizes were reciprocally similar at least 80%. Annotated simple repeats were also used to extend the range used to match SVs because similar variants might be placed in different position of an annotated VNTR. We built a network using these matches to identify connected component representing SV sites where similar alleles cluster. If the component formed a clique, i.e. all SVs matched all other SVs, the SV site contained alleles that differed by small variants (relative to the SV size). If not, the SV site contained alleles with significant differences, usually due to important repeat expansion or contraction at a VNTR site. For most analysis at the SV site level, we used the most frequent allele's information, such as boundaries or size. The corresponding scripts are documented in the repository (see Section [4.8.3](#)).

#### **4.9.7.6 Allele frequency patterns across populations**

We performed a principal component analysis of the individuals using the allele counts across all SV sites. The allele counts represent the number of non-reference alleles called in each individual at each SV locus.

For the MESA cohort, we compared the principal components derived from the SV allele counts to the principal components produced by the TOPMed consortium using short variants. Population information was not available for these individuals, so we used the top three principal components from our SV analysis to cluster them and explore population patterns further. We used hierarchical clustering (`ward.D` method in R's `hclust` function) and cut the

tree to obtain 6 clusters.

For the 1000 Genomes Project dataset, we used public information about the samples' super populations to color the PCA graphs and explore population patterns.

For each SV site, we compared the allele frequencies between the clusters/super populations. To illustrate global population differences, we computed the range of allele frequencies across clusters/super populations. This distribution was compared to an empirical null distribution produced by shuffling the population assignments. As population-centric metric, we contrasted the allele frequency at each SV site in a super population versus the median allele frequency across all super populations. Again, we used the null dataset to illustrate the enrichment in SVs whose frequency in the super population of interest is different from the global allele frequency. We highlight SVs with a frequency differing by more than 10% from the median frequency, a threshold at which only a few SVs are expected to show population patterns based on the permuted dataset. The corresponding scripts are documented in the repository (see Section 4.8.3).

#### 4.9.7.7 Comparison with existing large-scale SV catalogs

We compared the variants genotyped in this study with three SV catalogs that screened large populations:

1. The 1000 Genomes Project phase 3[84] (2,504 individuals) downloaded from [http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/integrated\\_sv\\_map/supporting/GRCh38\\_positions/ALL.wgs.mergedSV.v8.20130502.svs.genotypes.GRCh38.vcf.gz](http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/integrated_sv_map/supporting/GRCh38_positions/ALL.wgs.mergedSV.v8.20130502.svs.genotypes.GRCh38.vcf.gz)

2. gnomAD-SV[38] v2 (more than 14 thousand individuals), lifted over to GRCh38
3. SVPOP[8] (440 individuals) downloaded from [http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data\\_collections/hgsv\\_sv\\_discovery/working/20181025\\_EEE\\_SV-Pop\\_1/VariantCalls\\_EEE\\_SV-Pop\\_1/EEE\\_SV-Pop\\_1.ALL.genotypes.20181204.vcf.gz](http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/hgsv_sv_discovery/working/20181025_EEE_SV-Pop_1/VariantCalls_EEE_SV-Pop_1/EEE_SV-Pop_1.ALL.genotypes.20181204.vcf.gz).

These three public catalogs provide allele frequency estimate that we compared with the frequency of matched variants in our datasets. Variants were matched with weaker matching criteria than before to avoid over-estimating the amount of novel variants. We matched deletions with at least 30% reciprocal overlap and insertions located at less than 200 bp from each other and at least 30% reciprocally similar sizes. Annotated simple repeats were also used to extend the range used to match SVs. The corresponding scripts are documented in the repository (see Section 4.8.3).

#### 4.9.7.8 Fine-tuning structural variants with frequencies

The frequency pattern across the different alleles at a SV site can help identify the discard alleles that contain errors. We have found thousands of SV sites where one allele is much more frequent than the other. Our most stringent definition selected 6,175 SV sites where the major allele is the only one with a frequency higher than 1% while also being at least three times more frequent than any other alleles.

These alleles were indeed of better quality at the base level when we compared them to the GIAB catalog[94], a SV catalog of unprecedented base quality. Out of the 22 of those SV sites that were found in GIAB, 3 (13.6%) matched exactly in terms of start/end position and

size. We show that it is significantly more than expected using permutation. We permuted the frequency estimates 10,000 times and showed that, on average, only 2.9% of SV sites would match GIAB exactly by chance (p-value < 0.0001).

#### 4.9.7.9 Annotation of the SVs

The RepeatMasker and SimpleRepeat annotation downloaded from the UCSC Genome Browser FTP server. We considered a SV to overlap simple repeats if at least 50% of its region overlapped with the SimpleRepeat track or repeats from the *Simple\_repeat* class in the RepeatMasker annotation. Low-complexity regions were extracted from the RepeatMasker annotation. The SVs were overlapped with gene annotation from Genecode v35. Promoter regions were defined from 2 kbp up-stream to 200 bp downstream of the transcription start site. We annotated SVs that overlapped coding regions, UTRs, promoters or introns of protein-coding genes, in this order. This means that a SVs spanning a full gene is annotated as *coding*, or, for example, that a *promoter* SV overlapped neither coding nor untranslated regions. Because the gene expression in the GEUVADIS is from lymphoblastoid cell lines, we used candidate regulatory regions for GM12878 from the ENCODE project[86, 25] ([ENCFF590IMH](#)) when annotating SV-eQTLs. As before, a SV was preferentially annotated using the following order: coding, UTR, promoter, intron, regulatory region, intergenic region.

#### 4.9.7.10 Expression Quantitative Trait Locus discovery

We used gene expression provided by the GEUVADIS consortium[87]. We tested for cis-eQTLs using a 1 Mbp range around the gene's transcription start site using Matrix

eQTL[76]. First, all 445 samples with gene expression information were analyzed jointly. Additionally, we also analyzed the CEU, FIN, GBR, and TSI samples together, and the YRI population separately. In each analysis, we only tested SVs with a minor allele frequencies of at least 1% and at least two samples with non-reference alleles. The gene expression had been corrected for batch effects by the GEUVADIS consortium using PEER[87]. To facilitate comparison of the predicted effect sizes, the gene expression distribution was centered and standardized before eQTL testing using the RowStandardizeCentered function in Matrix eQTL. Covariates of the model included sex and the top four principal components derived from the SV genotypes. The p-values were corrected for multiple-testing using Benjamini-Hochberg correction. We report eQTLs with an adjusted p-value (false discovery rate) of 1%. The p-value distributions and QQ plots were inspected for quality control.

We tested enrichment of SV-eQTLs in gene families extracted from the “Families” dataset of the HUGO Gene Nomenclature Committee (HGNC) BioMart server (downloaded March 15, 2021 from <https://biomart.genenames.org/>). The 1,552 gene families were tested using a hypergeometric test comparing genes with eQTLs versus genes that were tested in the eQTL analysis. The p-values were corrected for multiple testing using the Benjamini-Hochberg method.

We annotated each SV-eQTL using the gene annotation (see above). SVs were split between SVs associated with increased or decreased gene expression. As baseline for the functional overlaps, we selected genes with similar sizes as the genes involved in eQTLs. We then selected all common SVs (allele frequency  $\geq 1\%$ ) and located at less than 1 Mbp of these genes.

Separate analysis of the four European-ancestry populations together, and the YRI

population alone, identified respectively 44 and 139 SVs where an association with the expression of protein-coding genes was detected only in the smaller analysis. Possible reasons for this include the 5% frequency cutoff imposed on SV-eQTL candidates (which produced different sets of eligible variants in the different groups), the limited statistical power of the test (as the absence of a finding of an SV-eQTL is not the same as the finding of an absence of an SV-eQTL), and interactions between the candidate SV-eQTLs and the frequencies of other alleles or environmental factors which vary between the populations. As expected, a number of these population-specific SV-eQTLs had shown strong inter-super-population frequency patterns (see Section 4.5.13): 19 out of the 44 CEU-FIN-GBR-TSI-specific SV-eQTLs and 10 out of the 139 YRI-specific SV-eQTLs showed such patterns.

We also performed a joint analysis that included the public SNV and indel calls generated at the New York Genome Center[14]. Phased SNVs and indels in the autosomes and with minor allele frequencies of at least 1% were used for the eQTL analysis. The same approach as described above was performed. We defined the lead eQTL as the most significant association in a gene.

To evaluate the enrichment of SVs as lead eQTLs, we permuted the p-values within each gene 10,000 times to compute an empirical null distribution for the rate of lead SV-eQTL. We also used permutation to test for an enrichment of SV-eQTLs or SNV/indel-eQTLs among highly expressed genes. Genes were split in 20 expression classes based on their quantiles. We then computed an empirical p-value for the proportion of eQTLs in each expression class using an empirical distribution derived from permuting the expression classes 10,000 times. The p-values were corrected for multiple testing using the Benjamini-Hochberg method.

#### 4.9.8 Supplementary figures

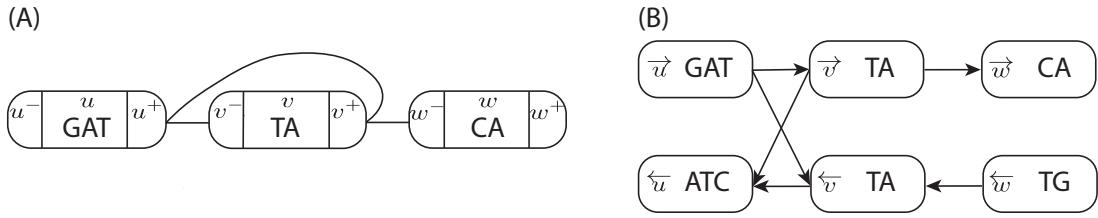


Figure 4.8: **Sequence graph example.** (A) A sequence graph with three nodes  $u$ ,  $v$ , and  $w$ , with labels  $\ell(u) = \text{GAT}$ ,  $\ell(v) = \text{TA}$ , and  $\ell(w) = \text{CA}$ . Node sides  $x^-$  and  $x^+$  are marked for each node  $x$ . (B) The same graph as a directed graph with each visit  $\vec{x}$  and  $\overleftarrow{x}$  a separate node.

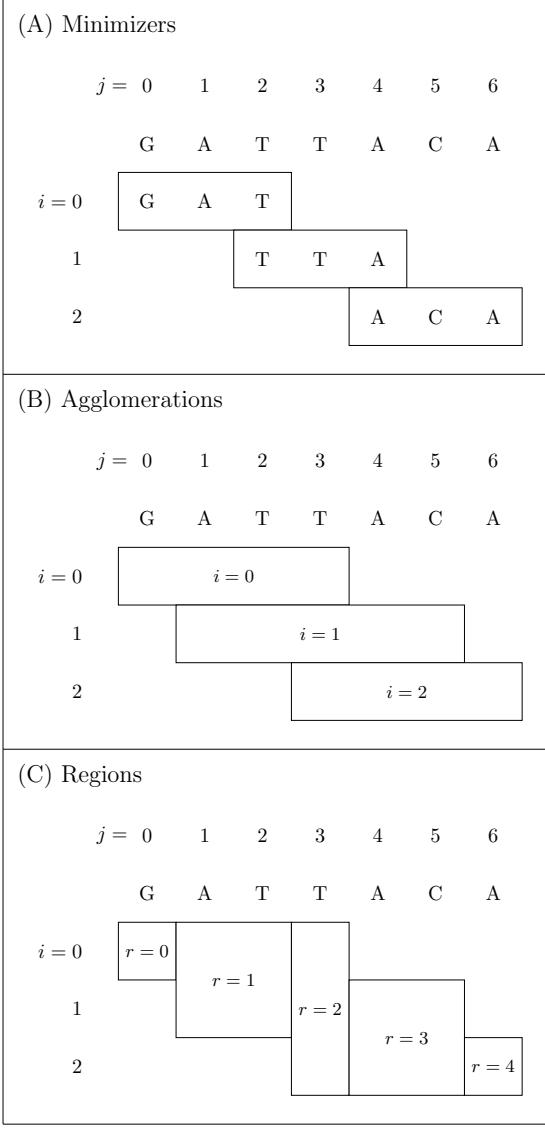
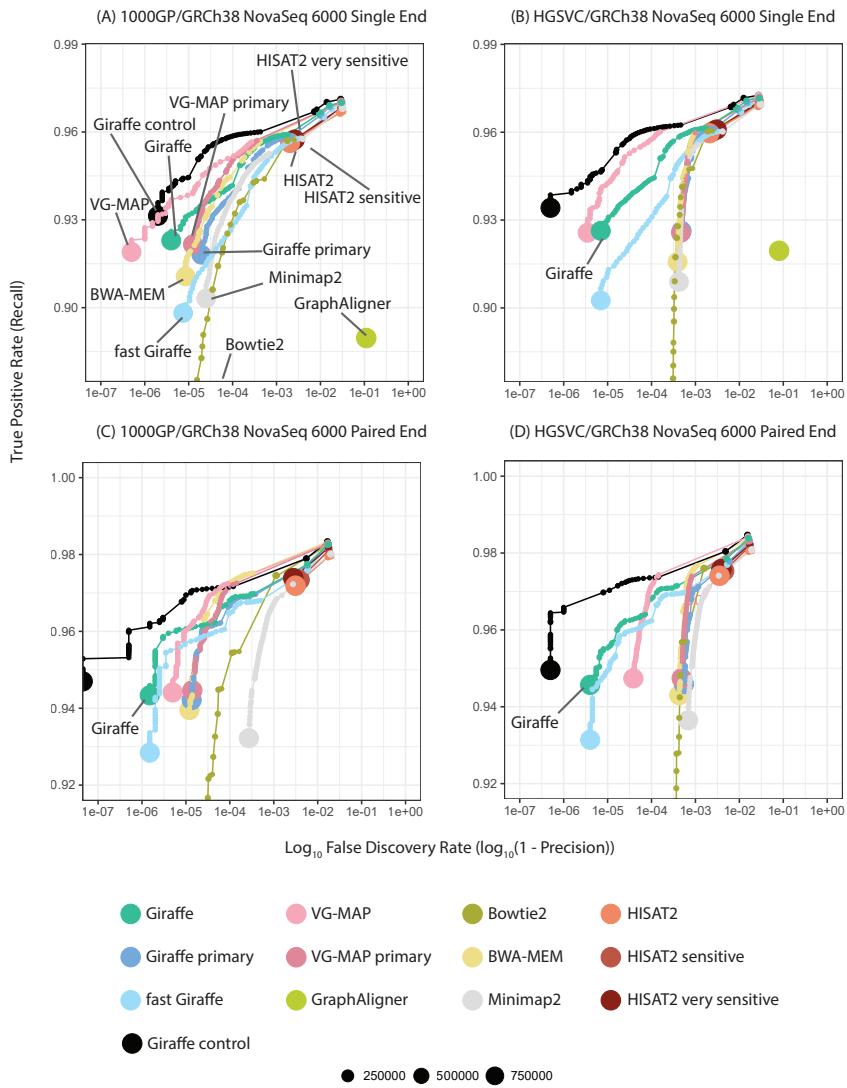
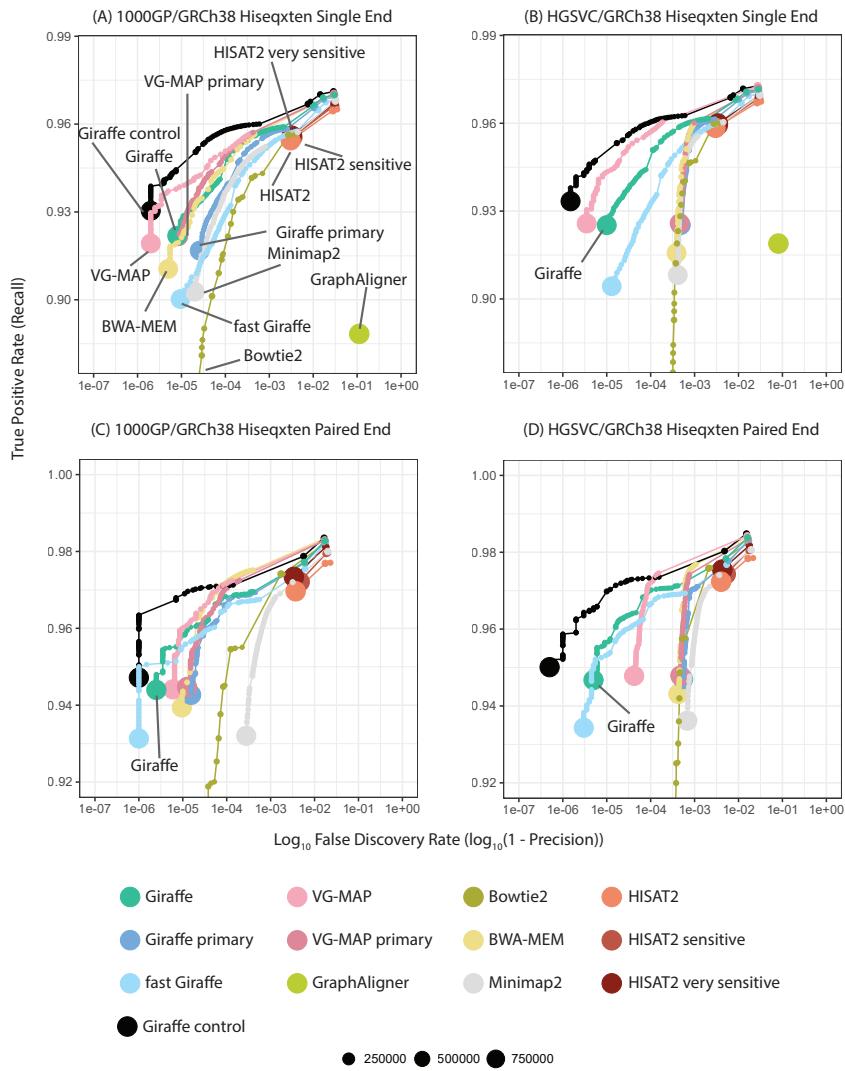


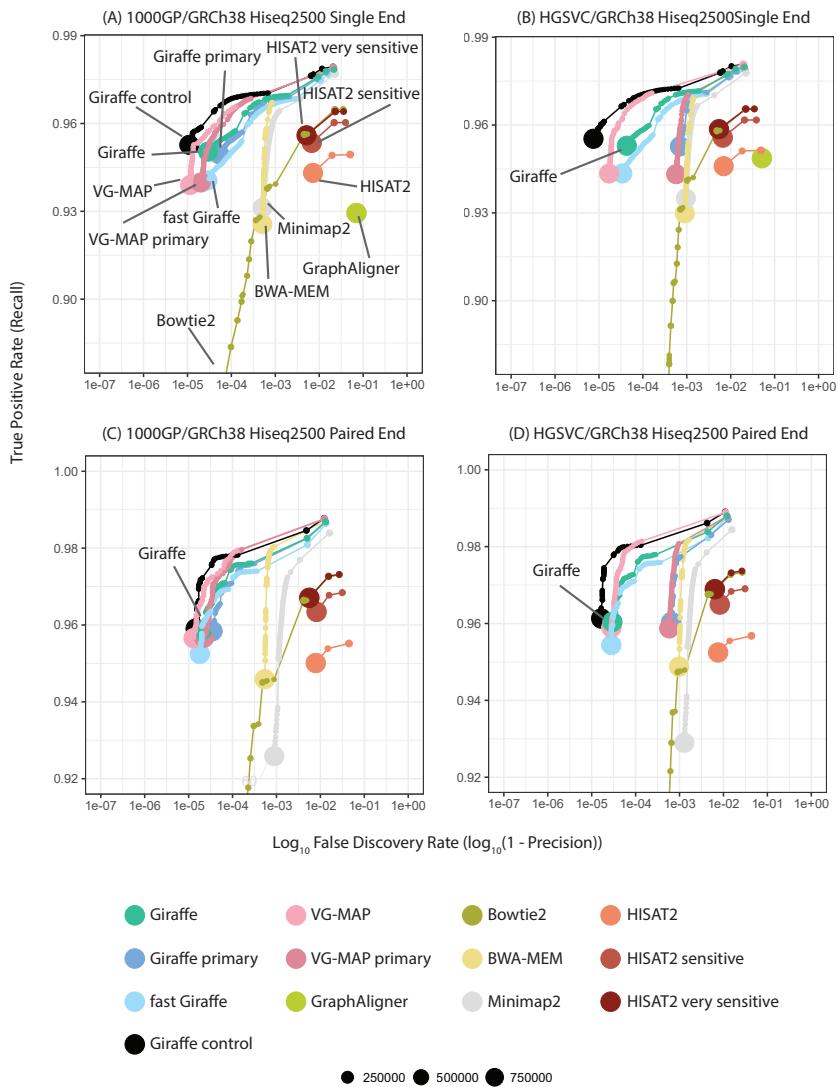
Figure 4.9: **Diagram of minimizers, agglomerations, and regions.** In this example, we use a minimizer length of  $k = 3$ , and a window size of  $w = 4$ . (A) shows where three example minimizers fall in the read. (B) shows the agglomerations of these minimizers, if the minimizers happen to be minimal for all windows they appear in. For agglomeration  $i = 1$ , we have  $\text{aggstart}(1) = 1$ ,  $\text{aggend}(1) = 5$ ,  $\text{minstart}(1) = 2$ , and  $\text{minend}(1) = 4$ . (C) shows the regions, which we use to decompose the part of the diagram covered by agglomerations, with breaks wherever an agglomeration begins or ends. For the region  $r = 3$ , we have  $\text{top}(3) = 1$ ,  $\text{bottom}(3) = 2$ ,  $\text{left}(3) = 4$ , and  $\text{right}(3) = 5$ .



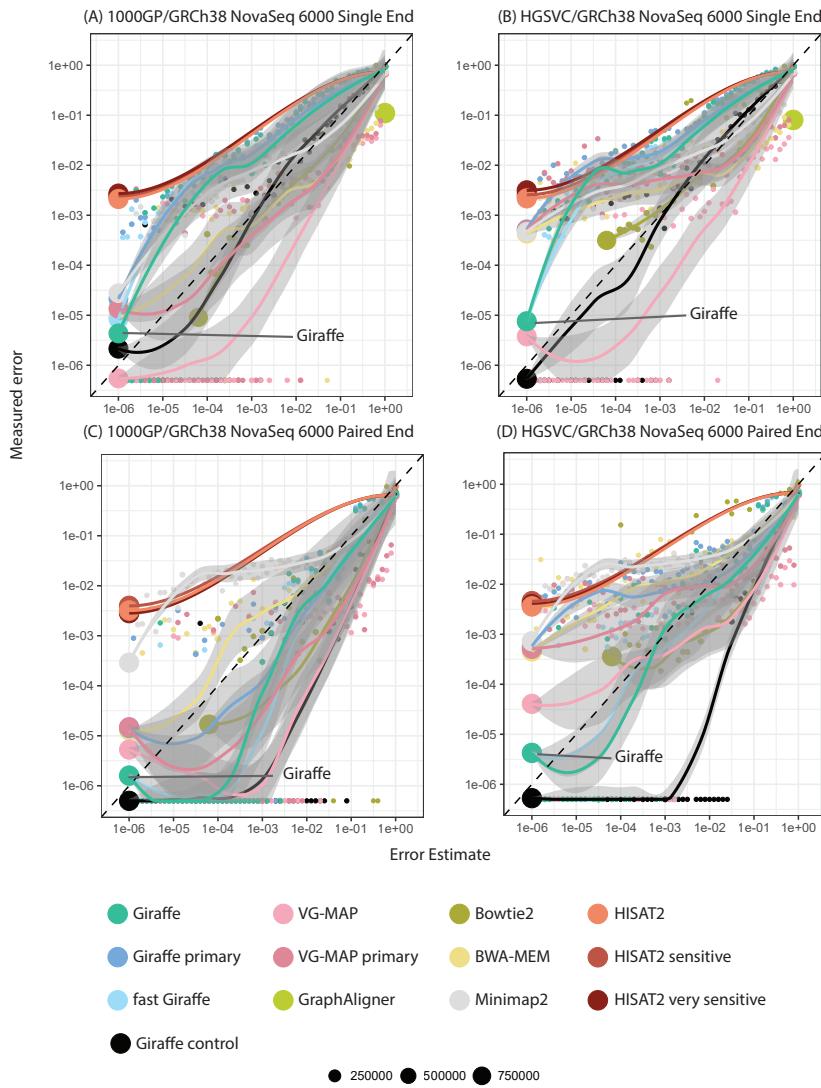
**Figure 4.10: Simulated read mapping with NovaSeq 6000 reads.** Each panel shows 1-precision (FDR) vs. recall for a simulated read mapping experiment. Reads were simulated to match 150bp Illumina NovaSeq 6000 reads and were mapped either as single ended reads (A,B) or as paired end reads (C,D). We compared graph mappers (Giraffe, VG-MAP, GraphAligner, HISAT2) and linear mappers (BWA-MEM, Bowtie2, Minimap2) mapping to either the 1000GP graph and GRCh38 reference (A,C) or the HGSVC graph and GRCh38 reference (B,D). For Giraffe, we mapped to a 64 haplotype sampled GBWT for the 1000GP graph and to the full GBWT for the HGSVC graph. In addition to aligning to the 1000GP and HGSVC graphs, we also used Giraffe and VG-MAP to align to primary graphs containing only the GRCh38 reference. We also ran Giraffe and HISAT2 using different pre-defined parameterizations in addition to the default settings: fast Giraffe and HISAT2 sensitive and very sensitive. For a positive control, we aligned reads to our positive control graphs using Giraffe run with a more exhaustive parameterization. Reads were stratified by their mapping quality. Each point in the plot represents to a mapping quality value and the size of a point corresponds the number of reads it represents. GraphAligner did not assign mapping qualities, so it is represented as a single point. For the remaining mappers, the mapping quality values varied from 0 to 60, except for Bowtie2, which had a maximum mapping quality of 42.



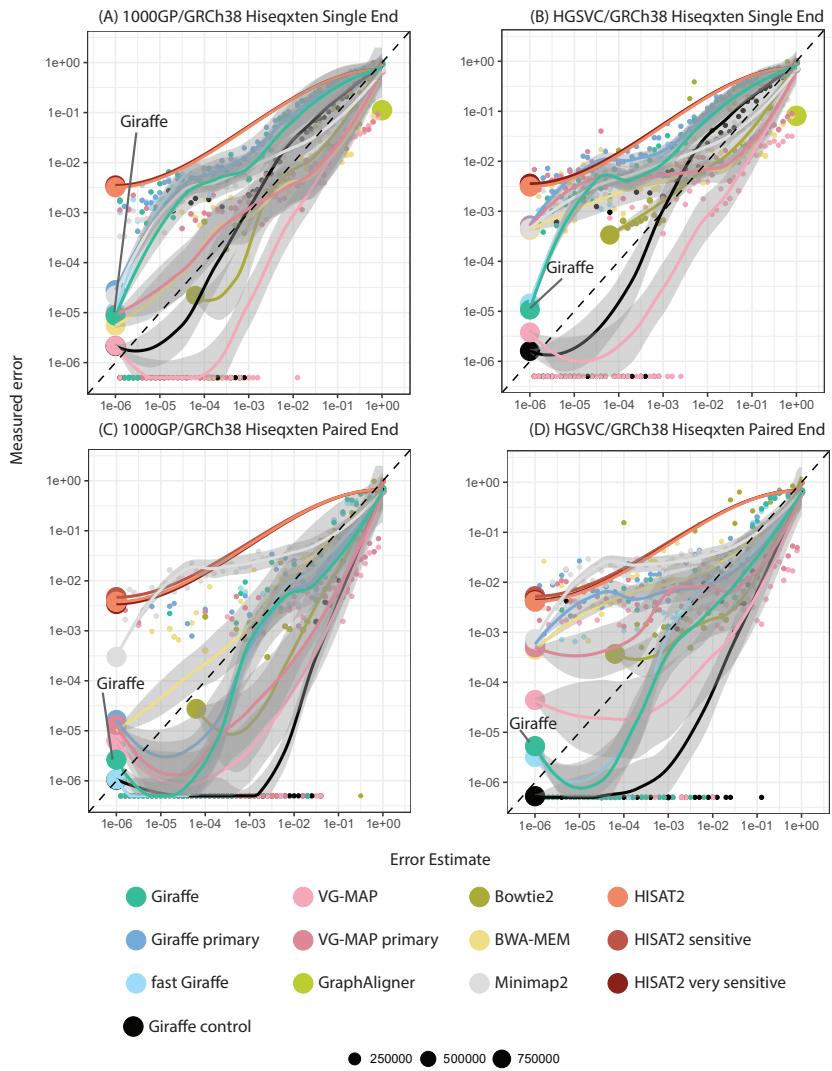
**Figure 4.11: Simulated read mapping with HiSeq X Ten reads.** Each panel shows 1-precision (FDR) vs. recall for a simulated read mapping experiment. Reads were mapped as single-end (A, B) or paired-end (C, D) to the 1000GP graph and GRCh38 reference (A, C) or the HGSVC graph and GRCh38 reference (B, D).



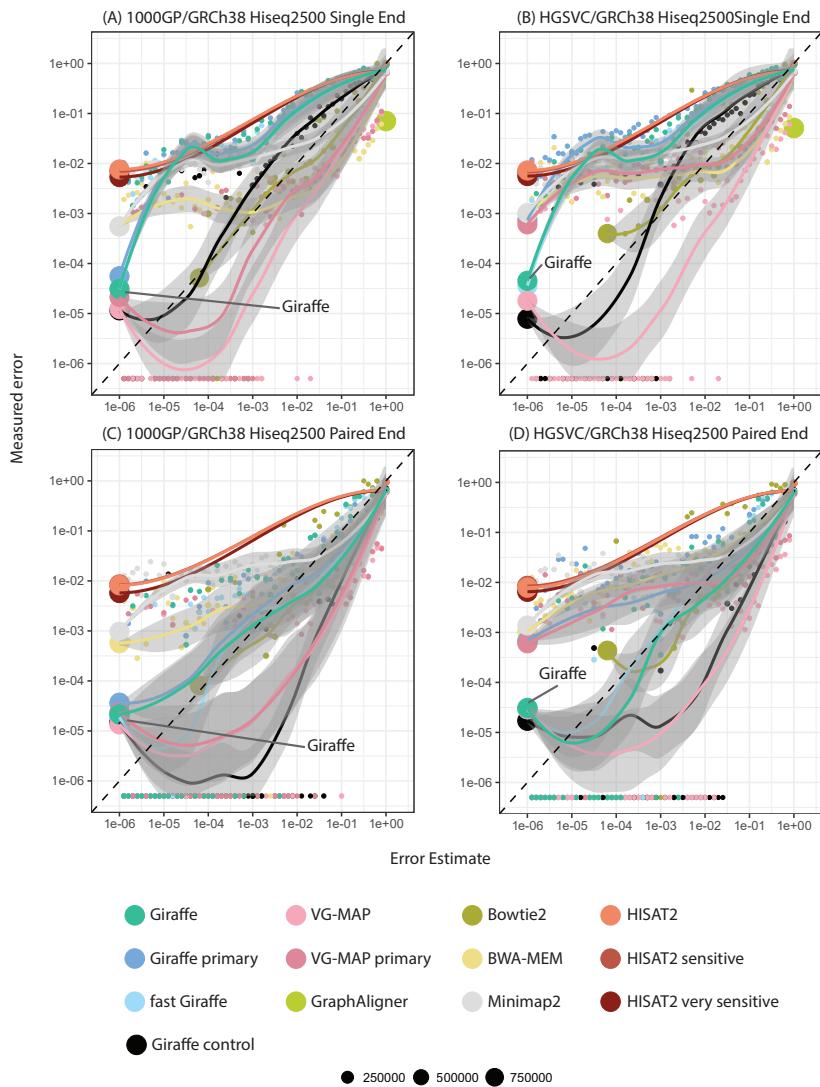
**Figure 4.12: Simulated read mapping with HiSeq 2500 reads.** Each panel shows 1-precision (FDR) vs. recall for a simulated read mapping experiment. Reads were mapped as single-end (A, B) or paired-end (C, D) to the 1000GP graph and GRCh38 reference (A, C) or the HGSVC graph and GRCh38 reference (B, D).



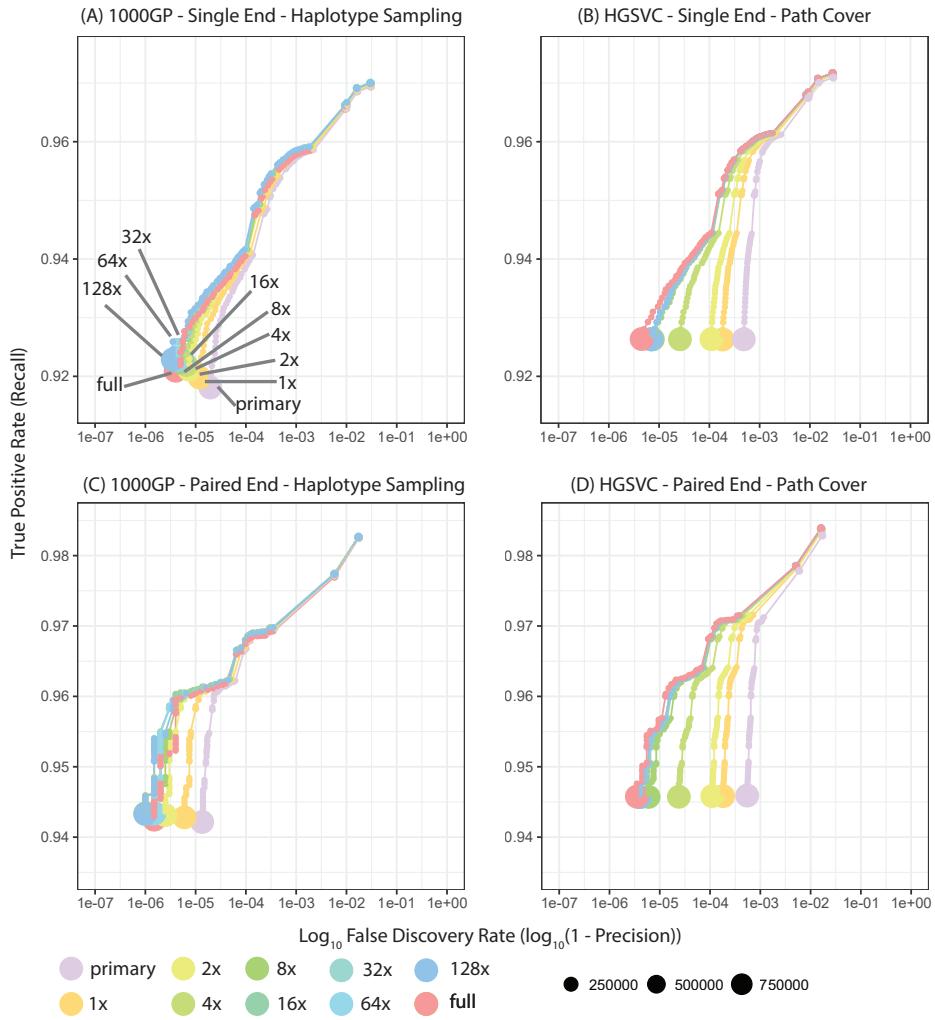
**Figure 4.13: QQ plot for simulated read mapping with NovaSeq 6000 reads.** Each panel shows a QQ plot for a simulated read mapping experiment. Reads were mapped as single-end (A, B) or paired-end (C, D) to the 1000GP graph and GRCh38 reference (A, C) or the HGSVC graph and GRCh38 reference (B, D).



**Figure 4.14: QQ plot for simulated read mapping with HiSeq X Ten reads.** Each panel shows a QQ plot for a simulated read mapping experiment. Reads were mapped as single-end (A, B) or paired-end (C, D) to the 1000GP graph and GRCh38 reference (A, C) or the HGSVC graph and GRCh38 reference (B, D).



**Figure 4.15: QQ plot for simulated read mapping with HiSeq 2500 reads.** Each panel shows a QQ plot for a simulated read mapping experiment. Reads were mapped as single-end (A, B) or paired-end (C, D) to the 1000GP graph and GRCh38 reference (A, C) or the HGSVC graph and GRCh38 reference (B, D).



**Figure 4.16: Assessing haplotype sampling and path cover.** Each panel shows recall vs. FDR for a simulated read mapping experiment. (A,C) Haplotype sampling, for samplings from 1 to 128 using the 1000GP derived graph (Table 4.8). Includes for comparison a graph containing just the primary reference (here GRCh38) and, separately, the full 1000GP GBWT. For both (A) single-ended and (C) paired-end mapping, performance saturates at around 64x coverage of sampled haplotypes, and exceeds that of mapping to the full GBWT containing all haplotypes. (B,D) Analogous path cover experiments, using the HGSVC graph (Table 4.7). All GBWT indexes were evaluated with simulated NovaSeq 6000 reads.

#### 4.9.9 Supplementary tables

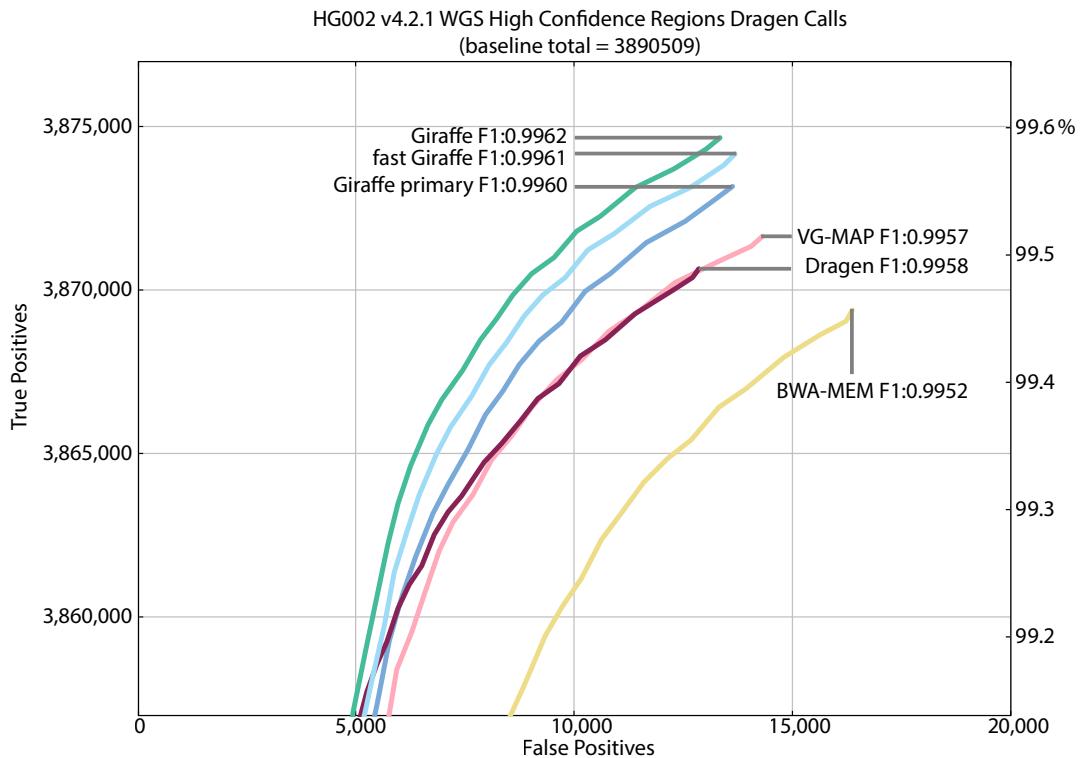


Figure 4.17: True positive and false positive genotypes made using the Dragen genotyper with projected mappings from Giraffe and other mappers, using 250bp paired-end reads from the HG002 GIAB sample and evaluated against the HG002 GIAB v4.2.1 truth variant call sets in high confidence regions. The ROC curve discrimination threshold is based on variant call quality.

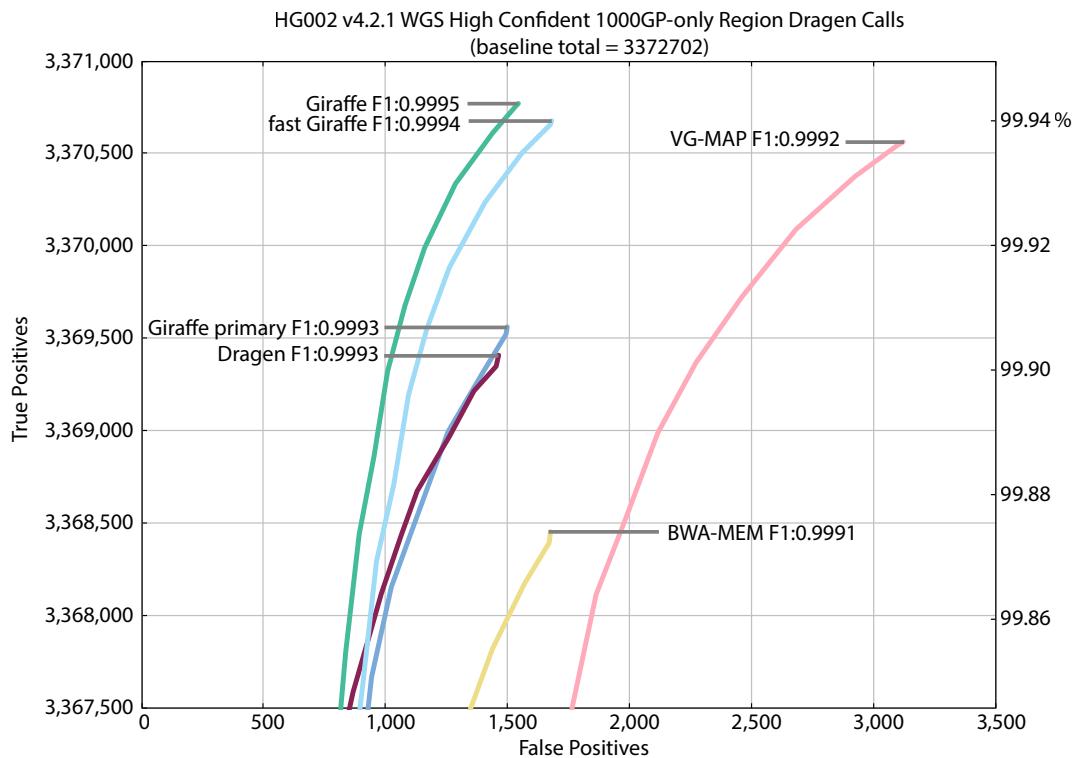


Figure 4.18: True positive and false positive genotypes made using the Dragen genotyper with projected mappings from Giraffe and other mappers, using 150bp paired-end reads from the HG002 GIAB sample and evaluated against the HG002 GIAB v4.2.1 truth variant call sets in high confidence regions only within 1000GP variant regions. The ROC curve discrimination threshold is based on variant call quality.

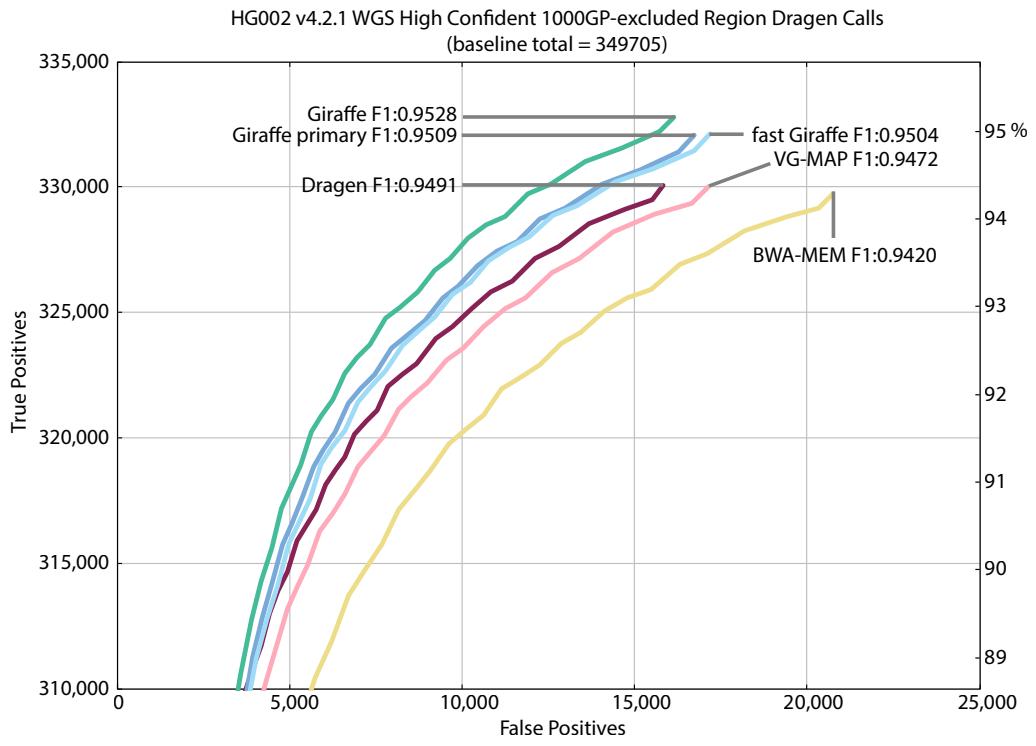


Figure 4.19: True positive and false positive genotypes made using the Dragen genotyper with projected mappings from Giraffe and other mappers, using 150bp paired-end reads from the HG002 GIAB sample and evaluated against the HG002 GIAB v4.2.1 truth variant call sets in high confidence regions with 1000GP variant regions excluded. The ROC curve discrimination threshold is based on variant call quality.

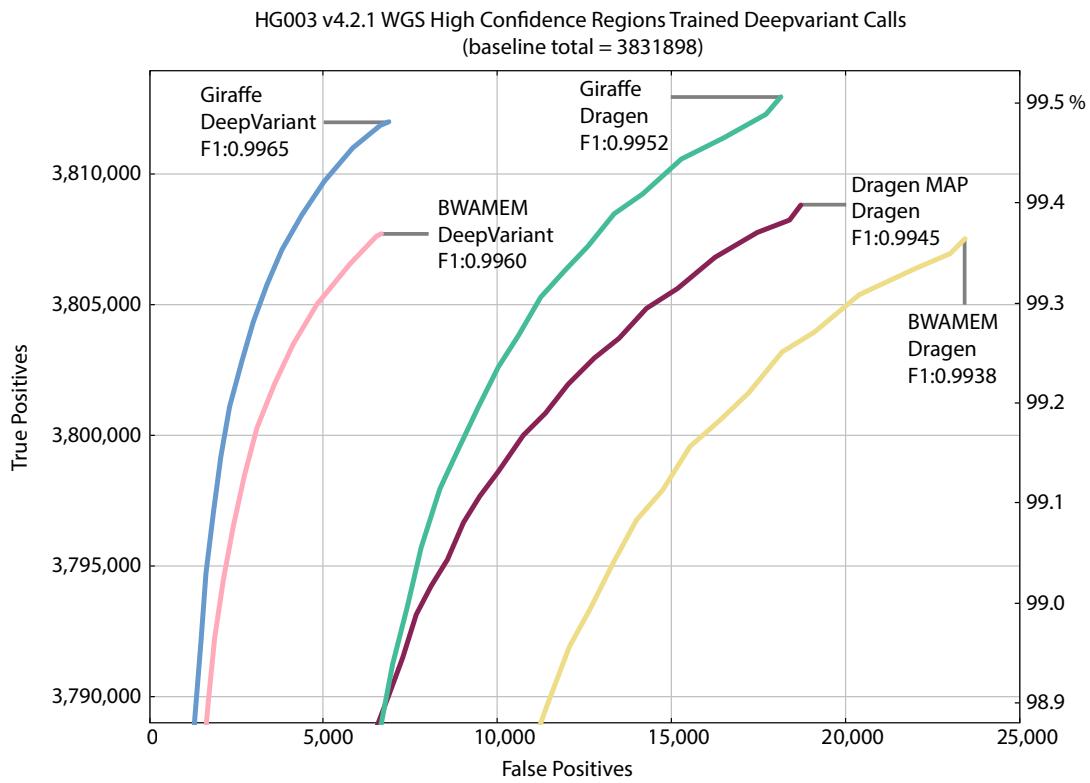
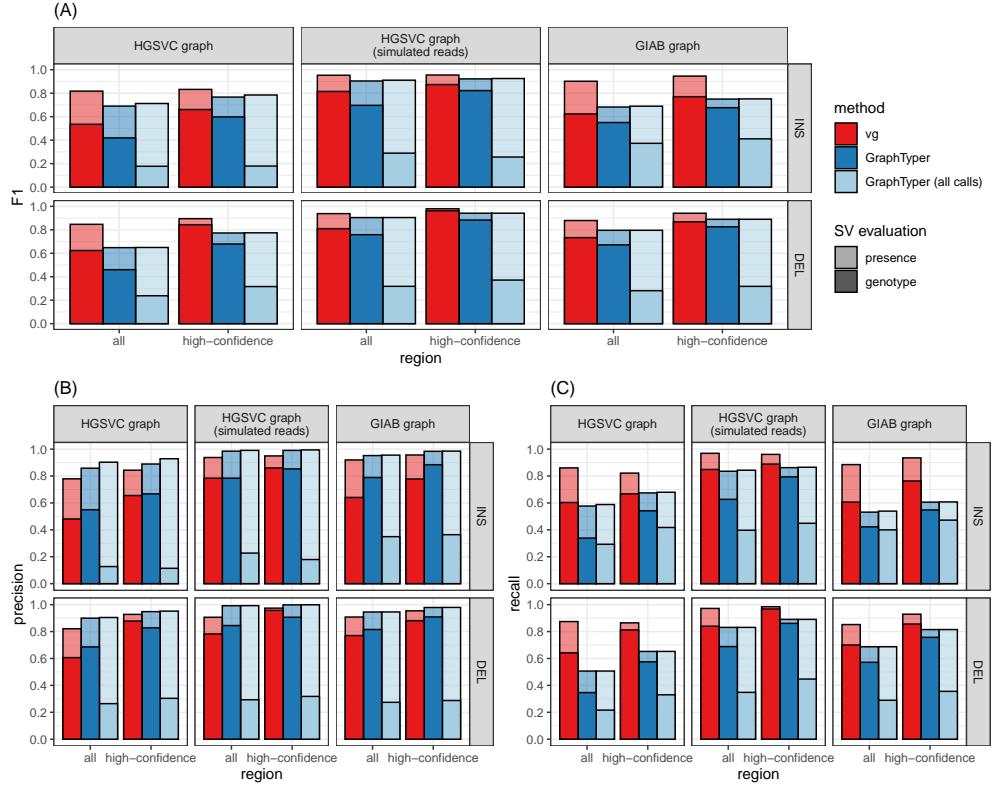


Figure 4.20: True positive and false positive genotypes made using the DeepVariant genotyper trained on alignments of 150bp paired-end reads from HG002 and HG004 GIAB samples and tested on 150bp paired-end reads from the HG003 GIAB sample and evaluated against the HG003 GIAB v4.2.1 truth variant call sets in high confidence regions. Dragen genotyper results are included for performance comparison. The ROC curve discrimination threshold is based on variant call quality.



**Figure 4.21: SV evaluation of **vg** and **GraphTyper** on HGSVC and GIAB.** SV were genotyped with **vg** and **GraphTyper** and compared to the truthset from HGSVC[15] and GIAB[94]. When genotyping the HGSVC catalog, we used short-reads from the three individuals and one set of simulated reads for one individual. When genotyping the GIAB catalog, we genotyped SV from short-read for HG002. The y-axis shows the F1 score when assessing the presence/absence of a SV (semi-transparent bars) or exact genotype (opaque bars), for **vg** (red) and **GraphTyper** (blue). **GraphTyper** outputs multiple entries for the same variant which decreases the genotyping accuracy when including all the variants (light blue). To improve the genotyping performance, we selected one prediction per variant for **GraphTyper** (darker blue). Finally, insertions and deletions were evaluated separately (top vs bottom) on the whole genome or high-confidence regions (x-axis). Panels A, B, and C show the F1 score, precision, and recall, respectively.

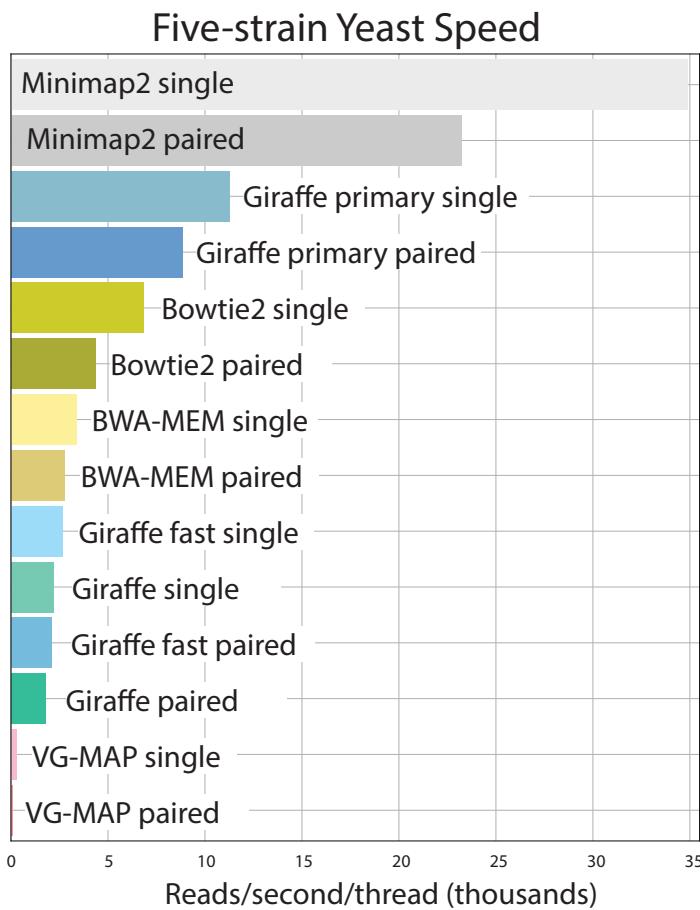
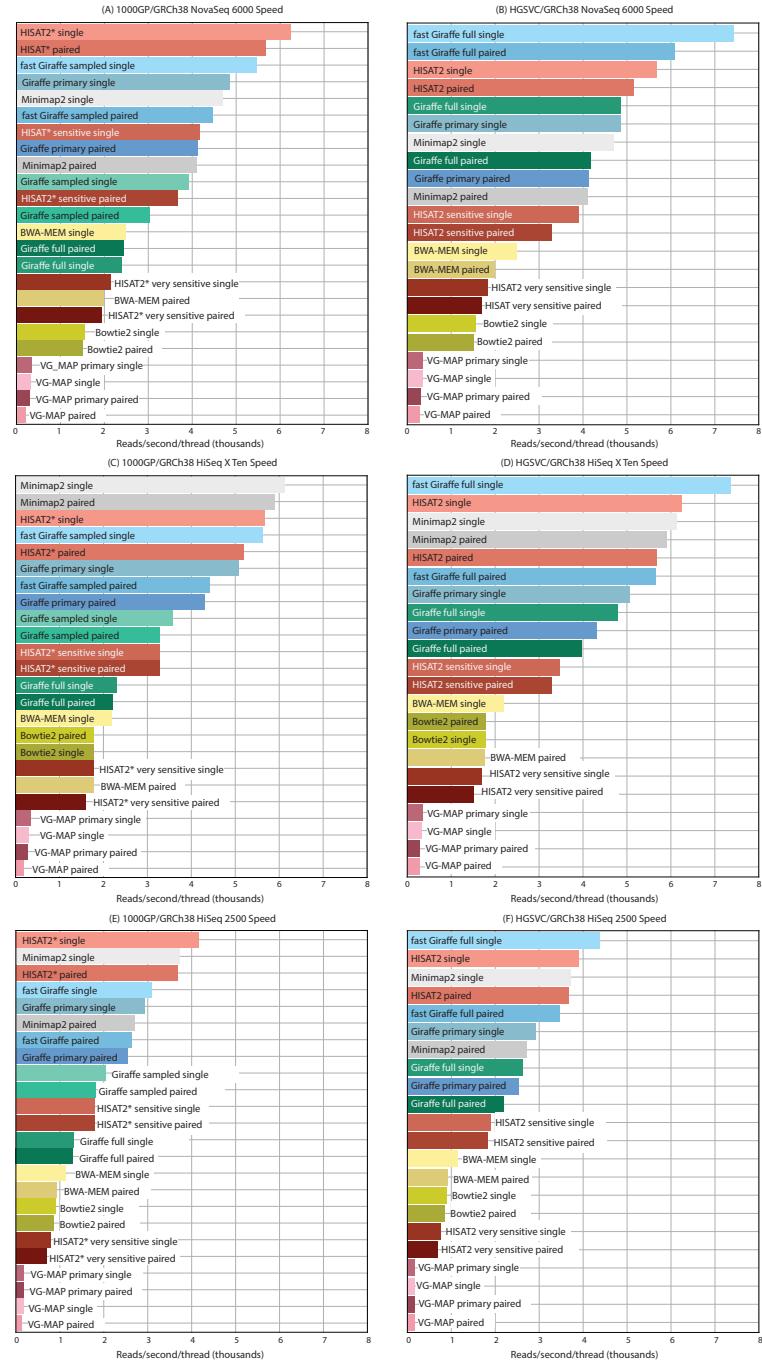
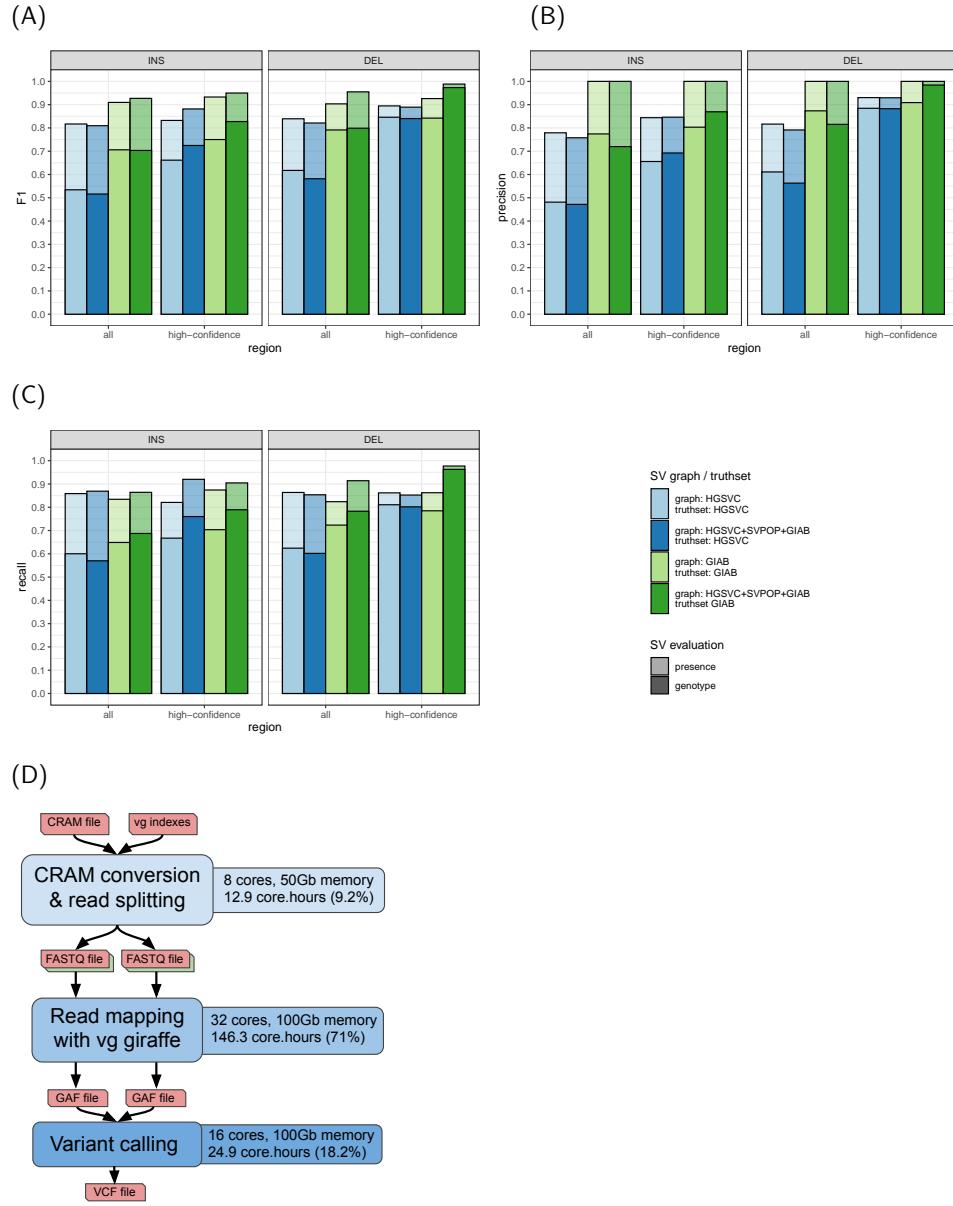


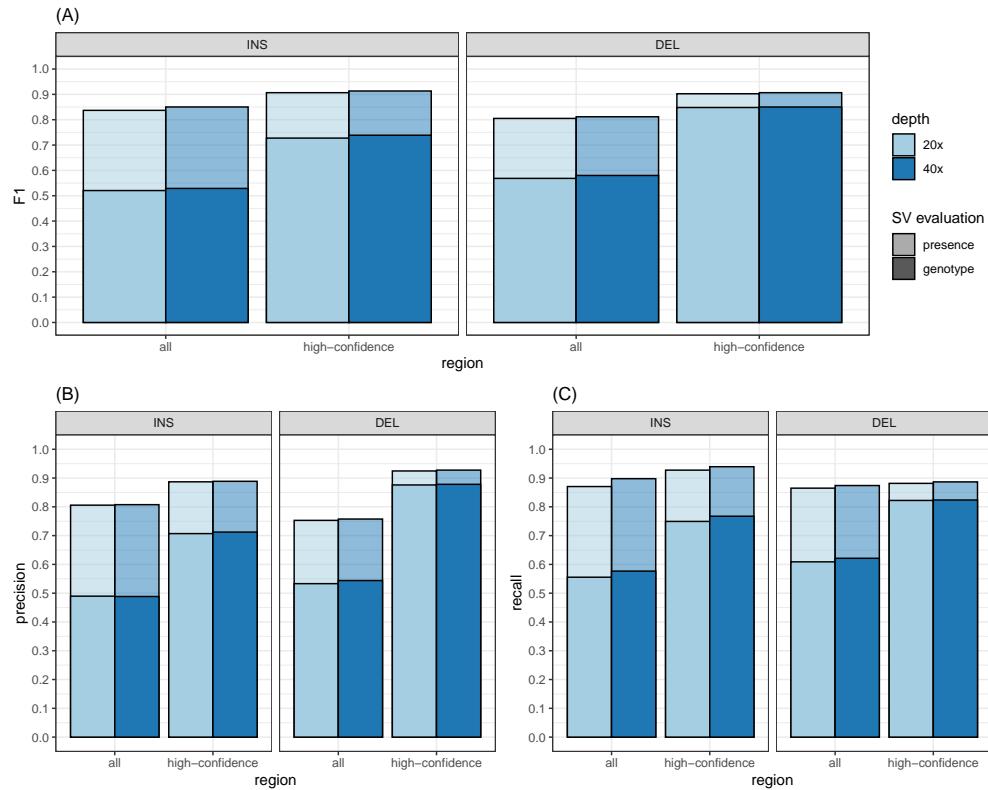
Figure 4.22: **Mapping speed on yeast data.** Each mapper was run on a dataset of 1 million real HiSeq 2500 reads from the DBVPG6044 strain on an AWS EC2 i3.8xlarge node with 32 vCPUs and 244GB of memory. The speed of mapping in reads per second per thread was determined using the total time spent mapping as reported by each tool. Each tool except Minimap2 was run on 16 threads; Minimap2 was run on 2 threads because it did not use all 16 threads.



**Figure 4.23: Mapping speed on human data.** Each mapper was run on a dataset of 1 million real NovaSeq 6000 (A,B), HiSeq X Ten (C,D), or HiSeq 2500 (E,F) reads on a AWS EC2 i3.8xlarge node with 32 vCPUs and 244GB of memory. The speed of mapping in reads per second per thread was determined using the total time spent mapping as reported by each tool. Each tool except Minimap2 was run on 16 threads; Minimap2 was run on 2 threads because it did not use all 16 threads.



**Figure 4.24: Structural variant genotyping workflow.** (A, B, C) SV genotyping F1, precision, and recall (respectively) for the SV graph combining HGSVC [15], SVPOP [8] and GIAB [94], and for individual SV graphs containing only HGSVC or GIAB as used in Hickey et al.[41]. The genotypes predicted from the long-read sequencing data in the original studies are used as truthset. (D) Workflow overview with average resources used for each task for a typical sample (Table 4.21). Note: the core hours estimates include the time spent copying files between computing instances.



**Figure 4.25: Minimal effect of down-sampling reads to 20x depth on the SV genotyping performance.** HG00514 was genotyped using the SV graph combining HGSVC, SVPOP and GIAB. Panels A, B, and C show the F1 score, precision, and recall, respectively.

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	96.94	91.81	0.00195
	VG-MAP primary	97.03	92.16	0.00125
	Bowtie2	96.84	82.85	0.00075
	BWA-MEM	97.05	91.07	0.00085
	Minimap2	96.82	90.32	0.00250
	Giraffe	97.00	92.29	0.00040
	fast Giraffe	96.86	89.82	0.00075
	VG-MAP	96.99	91.90	0.00005
	GraphAligner	88.95	-	-
	HISAT2	96.69	95.83	0.20275
	HISAT2 sens	96.76	95.92	0.22765
	HISAT2 vsens	96.83	96.00	0.25985
	Giraffe control	97.12	93.14	0.00020
paired	Giraffe Primary	98.25	94.22	0.00135
	VG-MAP Primary	98.28	94.47	0.00140
	Bowtie2	98.25	88.75	0.00150
	BWA-MEM	98.33	93.95	0.00120
	Minimap2	98.01	93.25	0.02710
	Giraffe	98.27	94.34	0.00015
	fast Giraffe	98.24	92.84	0.00015
	VG-MAP	98.29	94.41	0.00050
	HISAT2	97.95	97.50	0.31180
	HISAT2 sens	98.11	97.73	0.38640
	HISAT2 vsens	98.18	97.66	0.27780
	Giraffe control	98.33	94.70	0.00000

**Table 4.1: Mapping accuracy on NovaSeq 6000 reads mapped to the 1000GP graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	96.93	91.68	0.00260
	VG-MAP primary	97.04	92.17	0.00085
	Bowtie2	96.75	83.73	0.00185
	BWA-MEM	97.03	91.06	0.00050
	Minimap2	96.80	90.28	0.00200
	Giraffe	97.00	92.19	0.00080
	fast Giraffe	96.83	90.02	0.00095
	VG-MAP	97.00	91.92	0.00020
	GraphAligner	88.82	-	-
	HISAT2	96.51	95.75	0.30710
	HISAT2 sens	96.62	95.89	0.33675
	HISAT2 vsens	96.71	95.93	0.32535
paired	Giraffe control	97.11	93.04	0.00020
	Giraffe primary	98.27	94.27	0.00155
	VG-MAP primary	98.29	94.48	0.00125
	Bowtie2	98.21	89.07	0.00245
	BWA-MEM	98.33	93.94	0.00095
	Minimap2	98.00	93.23	0.02785
	Giraffe	98.28	94.40	0.00025
	fast Giraffe	98.24	93.13	0.00010
	VG-MAP	98.30	94.41	0.00060
	HISAT2	97.71	97.33	0.37060
	HISAT2 sens	97.95	97.66	0.45725
	HISAT2 vsens	98.12	97.68	0.33825
	Giraffe control	98.35	94.72	0.00010

Table 4.2: **Mapping accuracy on HiSeq X Ten reads mapped to the 1000GP graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	97.80	94.95	0.00530
	VG-MAP primary	97.90	93.99	0.00200
	Bowtie2	96.48	84.41	0.00435
	BWA-MEM	97.84	92.61	0.05120
	Minimap2	97.69	93.17	0.05120
	Giraffe	97.84	95.04	0.00295
	fast Giraffe	97.72	94.06	0.00285
	VG-MAP	97.90	93.90	0.00115
	GraphAligner	92.93	-	-
	HISAT2	94.94	95.03	0.72205
	HISAT2 sens	96.03	96.01	0.67535
	HISAT2 vsens	96.41	96.11	0.51385
paired	Giraffe control	97.94	95.27	0.00110
	Giraffe primary	98.68	95.84	0.00350
	VG-MAP primary	98.76	95.67	0.00225
	Bowtie2	97.30	87.87	0.00715
	BWA-MEM	98.71	94.64	0.05415
	Minimap2	98.39	92.68	0.08915
	Giraffe	98.69	95.87	0.00205
	fast Giraffe	98.65	95.25	0.00180
	VG-MAP	98.76	95.66	0.00130
	HISAT2	95.52	95.80	0.78575
	HISAT2 sens	96.84	97.16	0.81925
	HISAT2 vsens	97.32	97.27	0.56045
	Giraffe control	98.78	95.91	0.00145

Table 4.3: **Mapping accuracy on HiSeq 2500 mapped to the 1000GP graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	97.09	92.68	0.04840
	VG-MAP primary	97.17	92.62	0.04685
	Bowtie2	97.02	84.20	0.02645
	BWA-MEM	97.16	91.61	0.03925
	Minimap2	96.96	90.93	0.04220
	Giraffe	97.17	92.63	0.00070
	fast Giraffe	97.02	90.25	0.00070
	VG-MAP	97.24	92.57	0.00035
	GraphAligner	91.94	-	-
	HISAT2	97.10	96.36	0.20975
	HISAT2 sens	97.16	96.46	0.24702
	HISAT2 vsens	97.24	96.58	0.30368
paired	Giraffe control	97.26	93.42	0.00005
	Giraffe primary	98.29	94.65	0.05480
	HISAT2	96.51	95.75	0.30710
	Bowtie2	98.28	89.54	0.03220
	BWA-MEM	98.38	94.34	0.04275
	Minimap2	98.09	93.72	0.06800
	Giraffe	98.39	94.57	0.00040
	fast Giraffe	98.35	93.14	0.00040
	VG-MAP	98.41	94.74	0.00385
	HISAT2	98.24	97.95	0.35377
	HISAT2 sens	98.36	98.17	0.44951
	HISAT2 vsens	98.41	98.16	0.39059
	Giraffe control	98.47	94.96	0.00005

**Table 4.4: Mapping accuracy on NovaSeq 6000 reads mapped to the HGSVC graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	97.10	92.57	0.04850
	VG-MAP primary	97.20	92.63	0.04585
	Bowtie2	96.94	85.15	0.02865
	BWA-MEM	97.19	91.61	0.03885
	Minimap2	96.96	90.86	0.04140
	Giraffe	97.16	92.52	0.00100
	fast Giraffe	97.01	90.43	0.00130
	VG-MAP	97.27	92.58	0.00035
	GraphAligner	91.88	-	-
	HISAT2	96.92	96.32	0.30077
	HISAT2 sens	97.02	96.45	0.34180
	HISAT2 vsens	97.11	96.52	0.34185
paired	Giraffe control	97.27	93.34	0.00015
	Giraffe primary	98.29	94.76	0.05470
	VG-MAP primary	98.35	94.84	0.04750
	Bowtie2	98.25	89.98	0.03325
	BWA-MEM	98.40	94.36	0.04280
	Minimap2	98.07	93.69	0.06815
	Giraffe	98.38	94.68	0.00050
	fast Giraffe	98.34	93.43	0.00030
	VG-MAP	98.43	94.79	0.00420
	HISAT2	98.04	97.82	0.40337
	HISAT2 sens	98.23	98.13	0.51313
	HISAT2 vsens	98.36	98.16	0.42937
	Giraffe control	98.48	95.01	0.00005

**Table 4.5: Mapping accuracy on HiSeq X Ten reads mapped to the HGSVC graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	Giraffe primary	97.89	95.34	0.06980
	VG-MAP primary	97.98	94.38	0.05715
	Bowtie2	96.56	85.74	0.03405
	BWA-MEM	97.93	93.08	0.09025
	Minimap2	97.78	93.59	0.09525
	Giraffe	97.97	95.30	0.00430
	fast Giraffe	97.87	94.35	0.00340
	VG-MAP	98.08	94.35	0.00170
	GraphAligner	94.86	-	-0
	HISAT2	95.32	95.47	0.69207
	HISAT2 sens	96.36	96.41	0.66196
	HISAT2 vsens	96.73	96.58	0.54163
paired	Giraffe control	98.07	95.53	0.00075
	Giraffe primary	98.70	96.10	0.06465
	VG-MAP primary	98.79	95.93	0.05845
	Bowtie2	97.33	88.71	0.03860
	BWA-MEM	98.75	94.97	0.09730
	Minimap2	98.44	93.02	0.12805
	Giraffe	98.80	96.04	0.00295
	fast Giraffe	98.78	95.44	0.00280
	VG-MAP	98.89	95.89	0.00280
	HISAT2	95.86	96.18	0.75739
	HISAT2 sens	97.09	97.51	0.82688
	HISAT2 vsens	97.55	97.72	0.63802
	Giraffe control	98.91	96.13	0.00165

Table 4.6: **Mapping accuracy on HiSeq 2500 reads mapped to the HGSVC graph/GRCh38 reference** Each mapper was run on 2 million simulated reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60. \*Bowtie2 had a maximum mapping quality of 42. GraphAligner did not assign mapping quality

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	primary	97.09	92.68	0.04840
	1x	97.13	92.65	0.01840
	2x	97.15	92.64	0.01095
	4x	97.16	92.63	0.00260
	8x	97.17	92.63	0.00075
	16x	97.17	92.63	0.00070
	32x	97.17	92.63	0.00070
	64x	97.17	92.63	0.00070
	128x	97.17	92.63	0.00070
	full	97.17	92.64	0.00045
paired	primary	98.29	94.65	0.05480
	1x	98.35	94.60	0.01820
	2x	98.36	94.59	0.01105
	4x	98.38	94.58	0.00240
	8x	98.38	94.57	0.00060
	16x	98.39	94.57	0.00040
	32x	98.39	94.57	0.00040
	64x	98.39	94.57	0.00040
	128x	98.39	94.57	0.00040
	full	98.39	94.58	0.00035

Table 4.7: **Mapping accuracy using path cover GBWT** GBWTs were generated using the path cover for the HGSVC graph, with between 1 and 128 haplotypes. For comparison, GBWTs with only the primary GRCh38 reference and with the full HGSVC set were used. Giraffe was run with each GBWT on 2 million simulated NovaSeq 6000 reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60.

Pairing	Mapper	% Correct	% MAPQ 60	% Incorrect and MAPQ 60
single	primary	96.94	91.81	0.00195
	1x	96.97	91.98	0.00120
	2x	96.99	92.11	0.00075
	4x	97.00	92.19	0.00065
	8x	97.00	92.24	0.00050
	16x	97.00	92.27	0.00055
	32x	97.00	92.28	0.00045
	64x	97.00	92.29	0.00040
	128x	97.00	92.29	0.00035
	full	96.94	92.10	0.00040
paired	primary	98.25	94.22	0.00135
	1x	98.26	94.28	0.00060
	2x	98.26	94.31	0.00025
	4x	98.26	94.33	0.00010
	8x	98.26	94.34	0.00010
	16x	98.26	94.34	0.00015
	32x	98.26	94.34	0.00015
	64x	98.27	94.34	0.00015
	128x	98.26	94.33	0.00010
	full	98.25	94.25	0.00015

Table 4.8: **Mapping accuracy using haplotype sampled GBWT** GBWTs were generated by sampling the 1000GP haplotypes, sampling between 1 and 128 haplotypes. For comparison, GBWTs with only the primary GRCh38 reference and with the full 1000GP set were used. Giraffe was run with each GBWT on 2 million simulated NovaSeq 6000 reads and assessed for the percent of reads that were mapped correctly, the percent of reads that were assigned mapping quality 60, and the percent of reads that were incorrect and assigned mapping quality 60.

Pipeline	TP	FP	FN	Precision	Sensitivity	F-measure
BWA-MEM	3,866,030	22,519	24,392	0.9942	0.9937	0.9940
DRAGEN	3,867,353	<b>17,369</b>	23,071	<b>0.9955</b>	0.9941	0.9948
VG-MAP	3,868,494	20,393	21,929	0.9948	0.9944	0.9946
Giraffe primary	3,869,525	18,308	20,909	0.9953	0.9946	0.9950
Giraffe	<b>3,871,501</b>	17,787	<b>18,917</b>	0.9954	<b>0.9951</b>	<b>0.9953</b>
fast Giraffe	3,870,737	18,964	19,681	0.9951	0.9949	0.9950

Table 4.9: VCFeval performance of linear and graph-based pipelines against grch38-based references using 150bp paired-end reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.

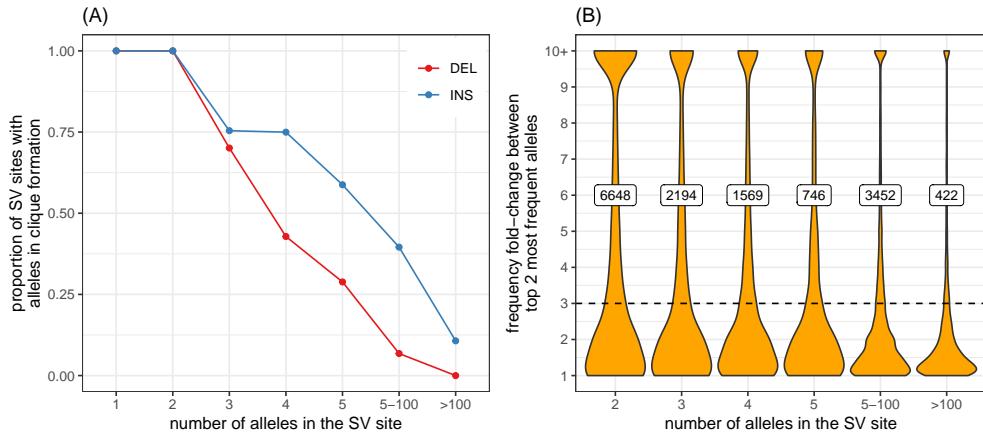
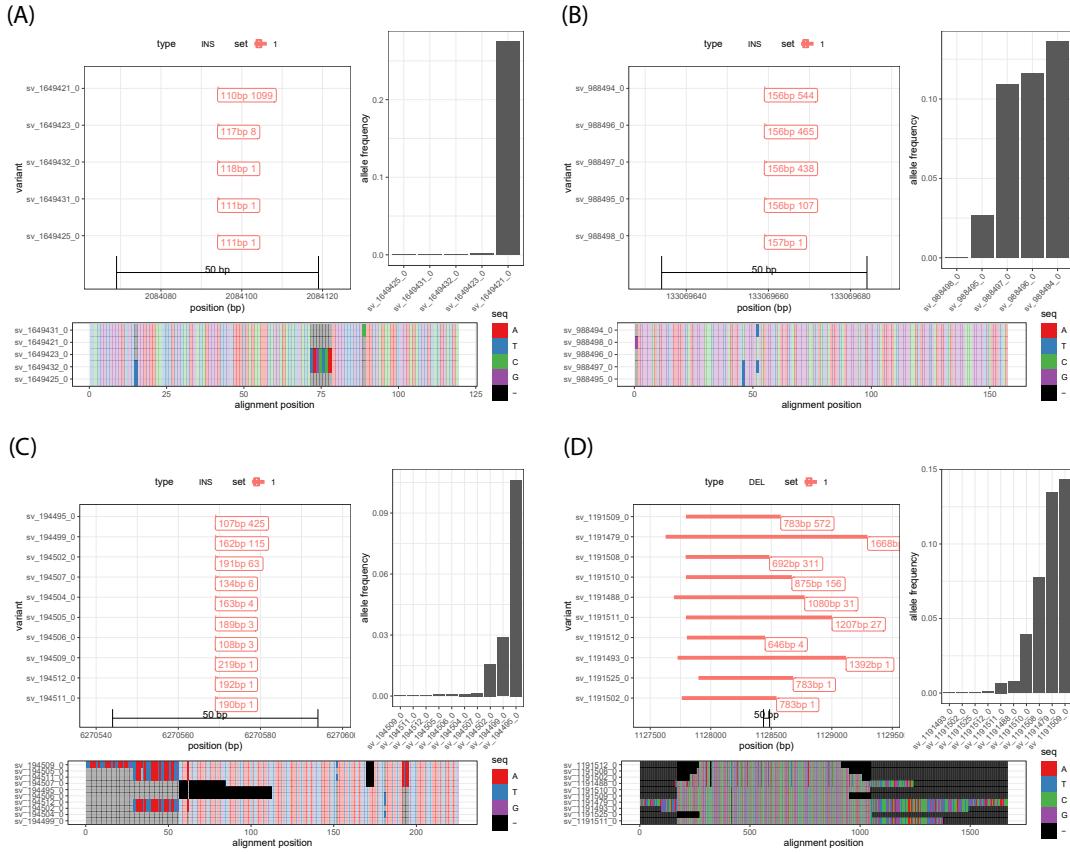


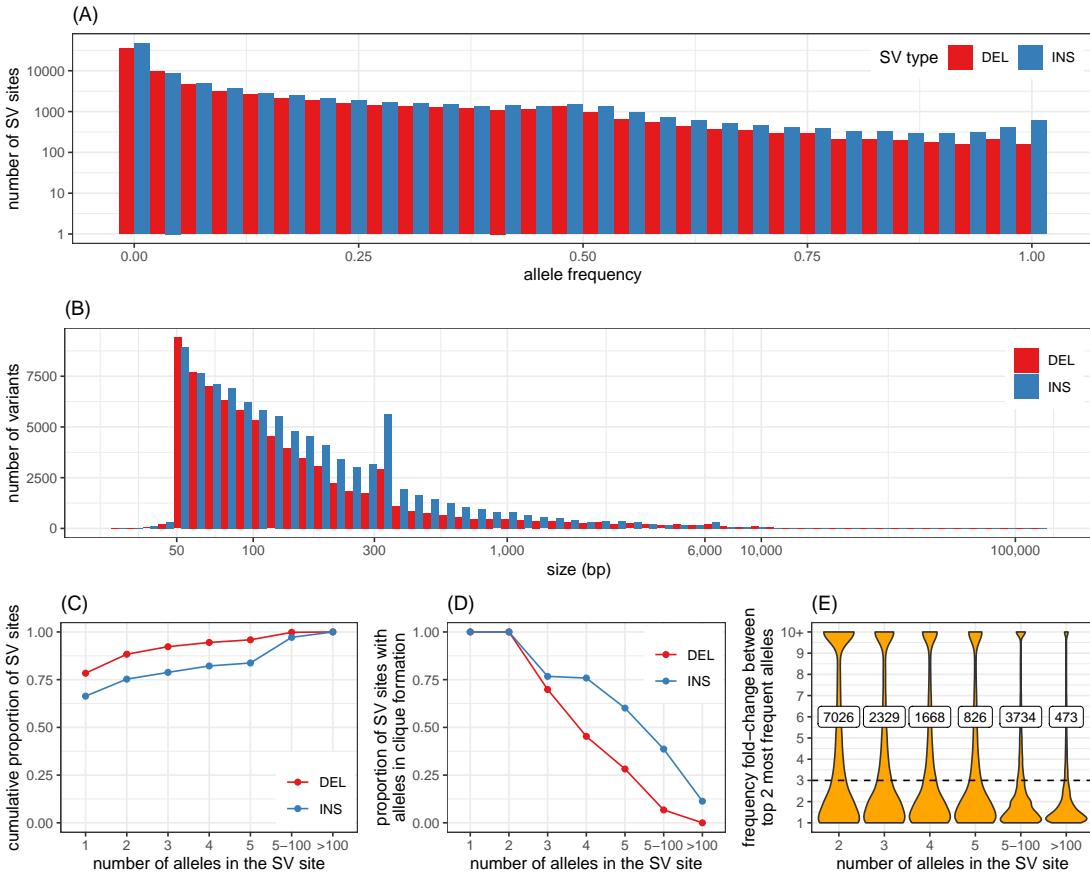
Figure 4.26: **SV alleles and SV sites in the MESA cohort.** (A) Proportion of SV loci in clique formation, i.e. with alleles different by only small variants (SNVs, indels) as opposed to large allelic variation such as VNTRs. The more alleles in a SV site the more likely it is to show VNTR-like allelic patterns. (B) Fold-change between the major allele and the second most frequent allele in a SV loci (y-axis), grouped by the number of alleles in the locus (x-axis). The number of loci with a fold-change higher than 3 (dotted line) are labeled.

Pipeline	Var Type	TP	FN	FP	Recall	Precision	F1
BWA-MEM	INDELS	522,406	3,063	2,723	0.994171	0.995028	0.994599
	SNPS	3,343,857	21,270	19,780	0.993679	0.994121	0.993900
DRAGEN	INDELS	522,618	2,851	<b>2,478</b>	0.994574	<b>0.995475</b>	0.995025
	SNPS	3,344,971	20,156	<b>14,868</b>	0.994010	<b>0.995576</b>	0.994793
VG-MAP	INDELS	522,643	2,826	2,689	0.994622	0.995092	0.994857
	SNPS	3,346,085	19,042	17,677	0.994341	0.994746	0.994544
Giraffe primary	INDELS	522,657	2,812	2,608	0.994649	0.995239	0.994944
	SNPS	3,347,097	18,030	15,672	0.994642	0.995341	0.994991
Giraffe	INDELS	<b>522,864</b>	<b>2,605</b>	2,550	<b>0.995043</b>	0.995346	<b>0.995194</b>
	SNPS	<b>3,348,882</b>	<b>16,245</b>	15,213	<b>0.995173</b>	0.995479	<b>0.995326</b>
fast Giraffe	INDELS	522,825	2,644	2,583	0.994968	0.995286	0.995127
	SNPS	3,348,156	16,971	16,358	0.994957	0.995139	0.995048

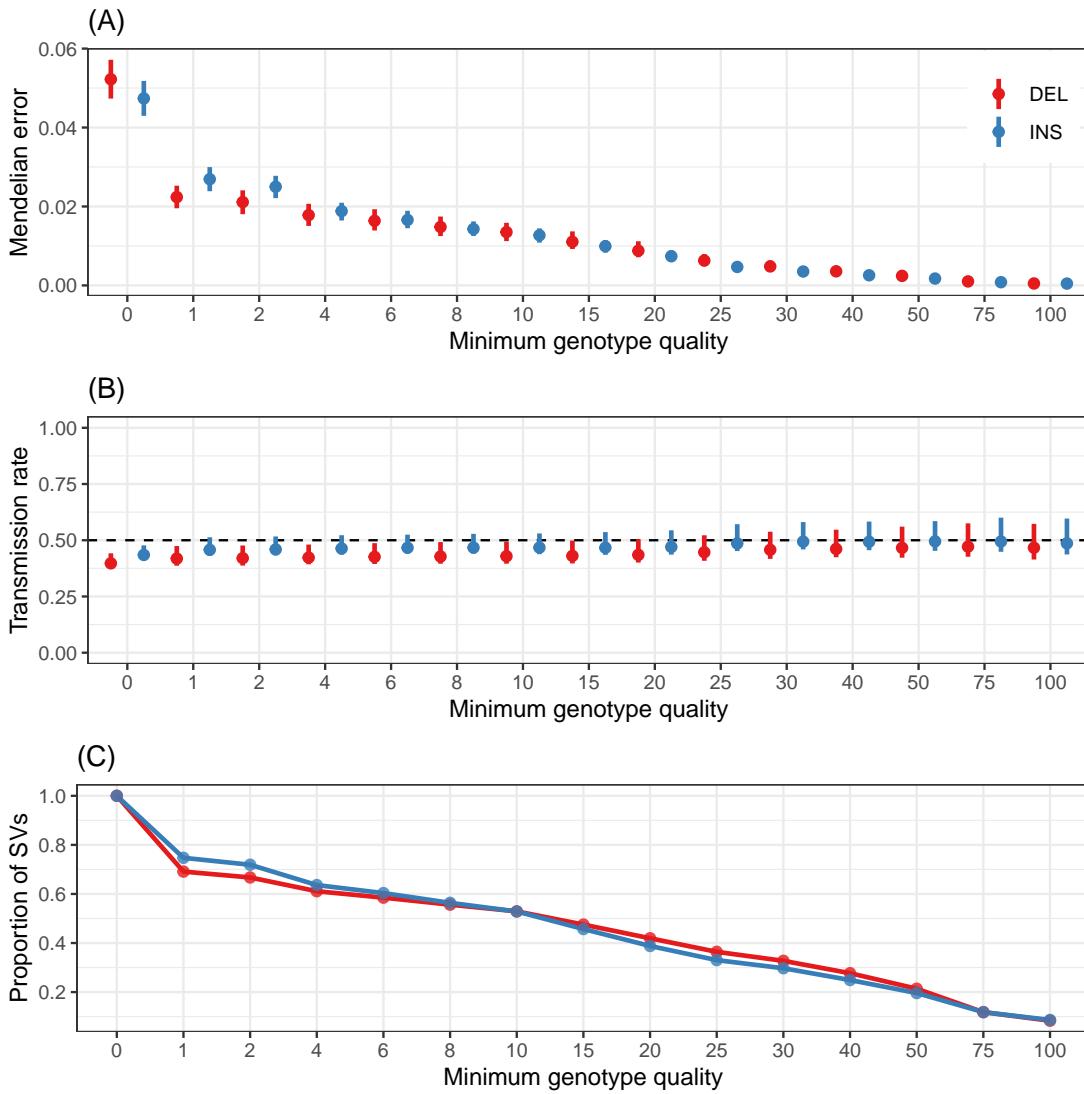
Table 4.10: Hap.py performance of linear and graph-based pipelines against grch38-based references using 150bp paired-end reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.



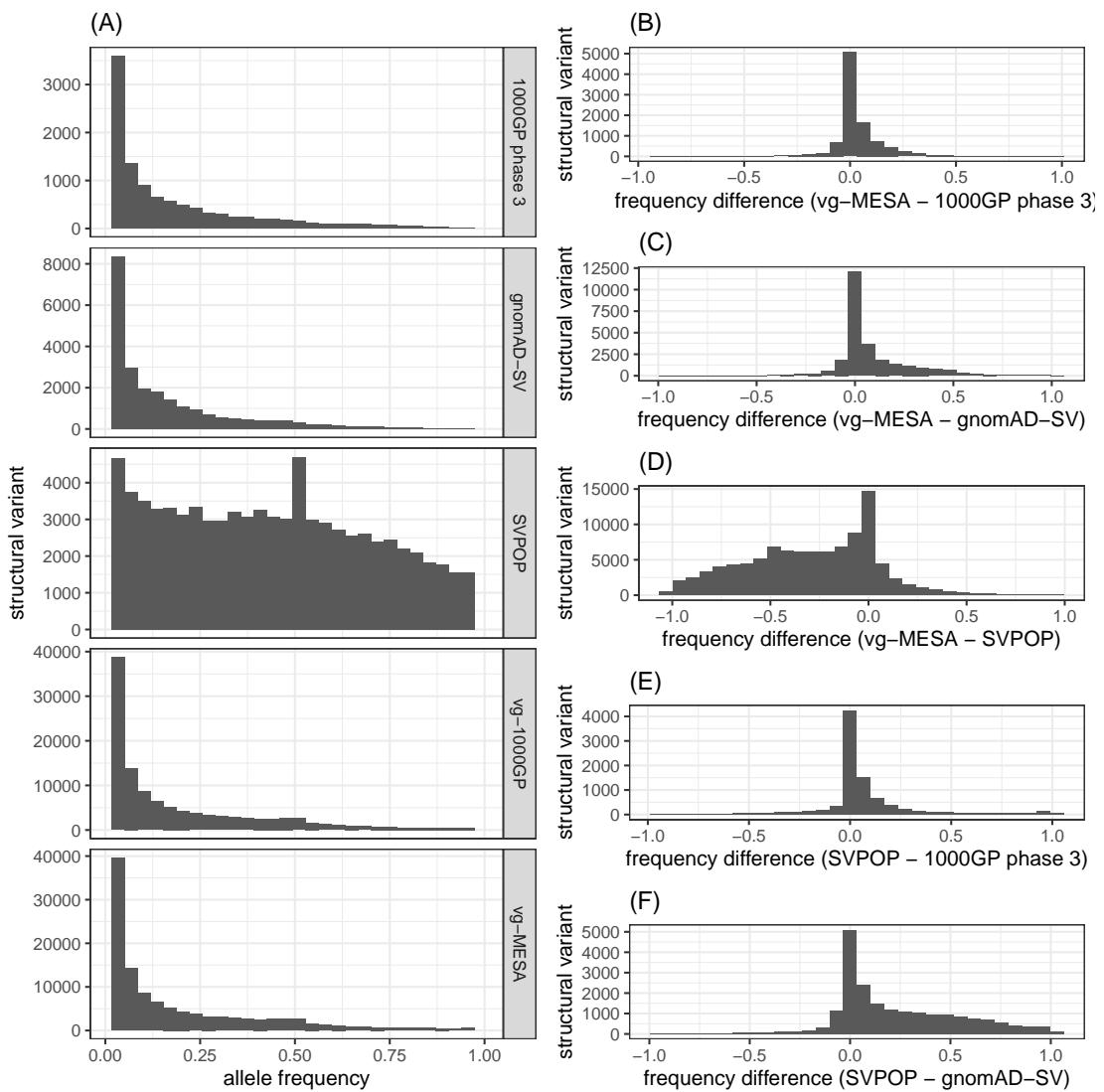
**Figure 4.27: Four examples of SV sites in the MESA cohort.** For each site, the top left panel shows a representation of their configuration in the genomic region, the top right panel shows the distribution of allele frequencies across the 2000 MESA samples, and the bottom panel shows the result of a multiple-sequence alignment of the inserted/deleted sequences. In (A) and (B), the insertions' alleles are only different due to small variants while (C) and (D) shows an insertion and deletion site with significant size variation between the alleles. In (A), there is one major allele while multiple alleles of (B) are frequent in the population.



**Figure 4.28: Structural variants in 2,504 unrelated individuals from the 1000 Genomes Project.** (A) Allele frequency distribution of the major allele for each SV loci. (B) Size distribution of the major allele for each SV loci. (C) Cumulative proportion of SV loci depending on the maximum number of alleles (x-axis) in the locus. (D) Proportion of SV loci in clique formation, i.e. with alleles different by only small variants (SNVs, indels) as opposed to large allelic variation such as VNTRs. The more alleles in a SV site the more likely it is to show VNTR-like allelic patterns. (E) Fold-change between the major allele and the second most frequent allele in a SV loci (y-axis), grouped by the number of alleles in the locus (x-axis). The number of loci with a fold-change higher than 3 (dotted line) are labeled.



**Figure 4.29: SV evaluation using 602 trios from the 1000 Genomes Project** (A) Mendelian error (y-axis) for different minimum genotype quality thresholds (x-axis), i.e. considering all variants where the minimum genotype quality in the trio is X or higher. (B) Proportion of transmitted allele from an heterozygous parent (y-axis) for the same minimum genotype quality thresholds (x-axis). In (A) and (B), the vertical lines represent the full error range across all trios while the point represents the median error. (C) Proportion of variants (y-axis) passing each minimum genotype quality threshold (x-axis).



**Figure 4.30: Allele frequency of SVs in this study and public studies.** (A) Allele frequency distribution of common variants (frequency  $\geq 1\%$ ) in each dataset. (B-D) show the difference in frequency between our results in the MESA cohort and each of the three public catalogs (1000 Genome Project phase 3, gnomAD-SV, SVPOP). (E-F) show the difference in frequency between SVPOP and the two other public catalogs.

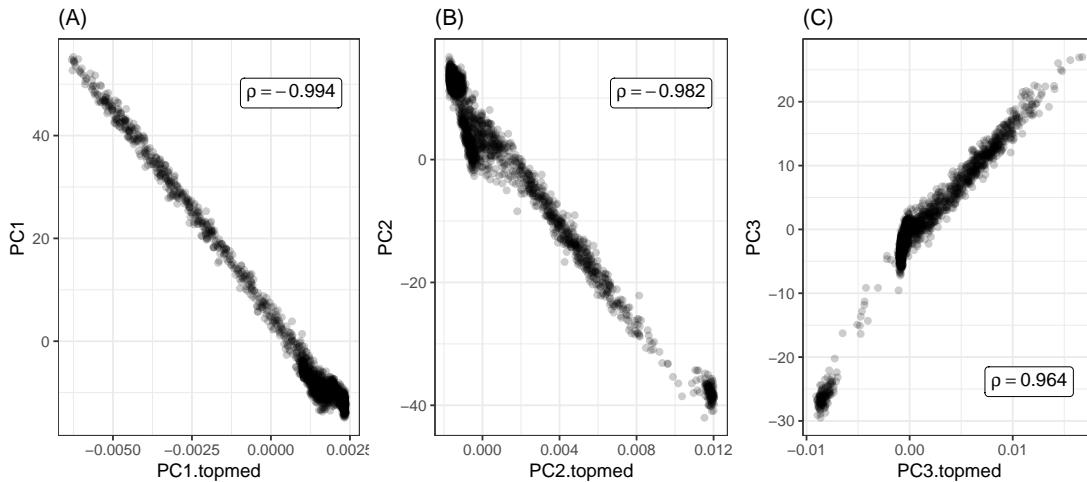
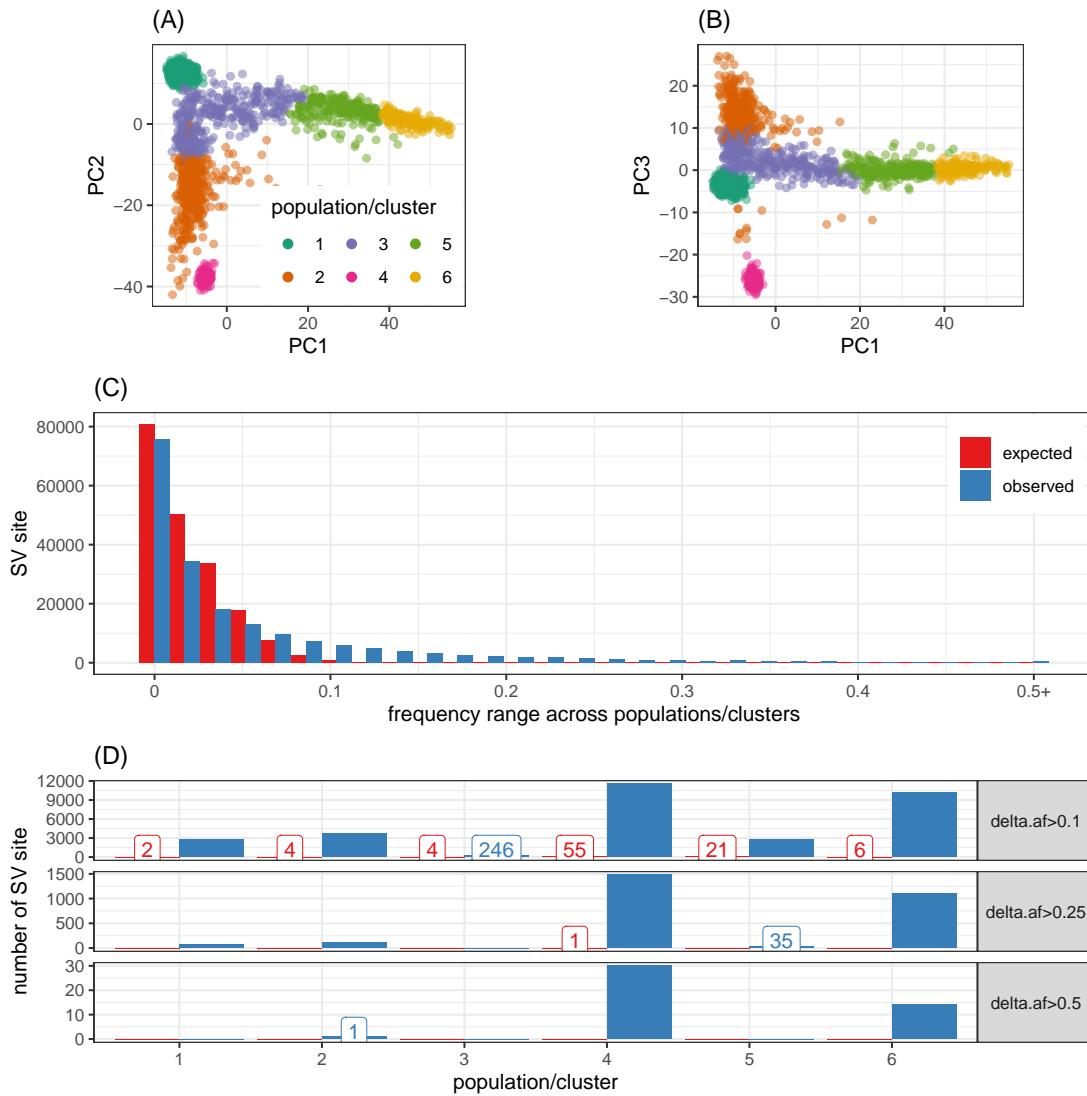


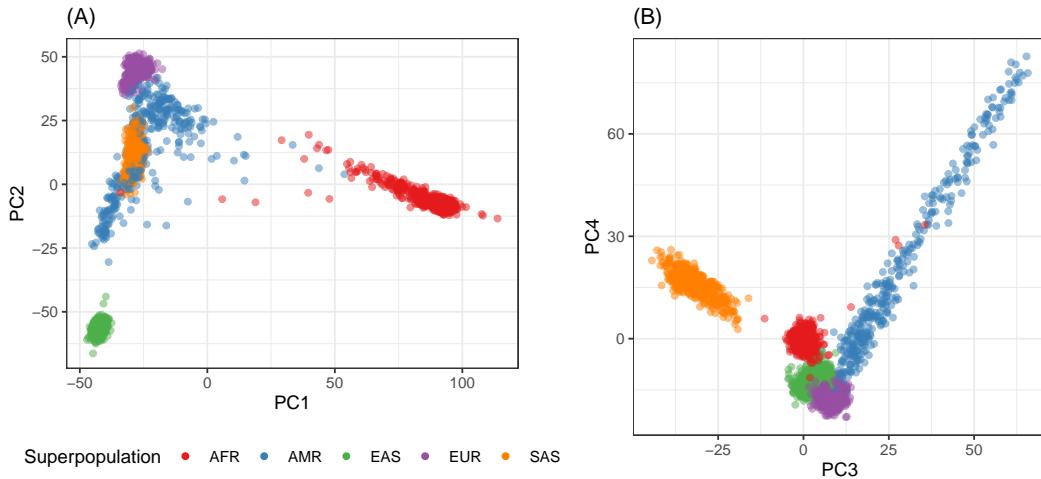
Figure 4.31: Principal component analysis using the SV genotypes (y-axis) in the MESA dataset compared to the principal components derived from TOPMed-wide SNVs (x-axis). (A), (B), (C) compare the first, second and third components.

Pipeline	TP	FP	FN	Precision	Sensitivity	F-measure
BWA-MEM	3,869,299	16,380	21,134	0.9958	0.9946	0.9952
DRAGEN	3,870,566	<b>12,863</b>	19,864	<b>0.9967</b>	0.9949	0.9958
VG-MAP	3,871,583	14,332	18,859	0.9963	0.9952	0.9957
Giraffe primary	3,873,118	13,640	17,331	0.9965	0.9955	0.9960
Giraffe	<b>3,874,597</b>	13,359	<b>15,840</b>	0.9966	<b>0.9959</b>	<b>0.9962</b>
fast Giraffe	3,874,104	13,695	16,334	0.9965	0.9958	0.9961

Table 4.11: VCFeval performance of linear and graph-based pipelines against grch38-based references using 250bp paired reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.



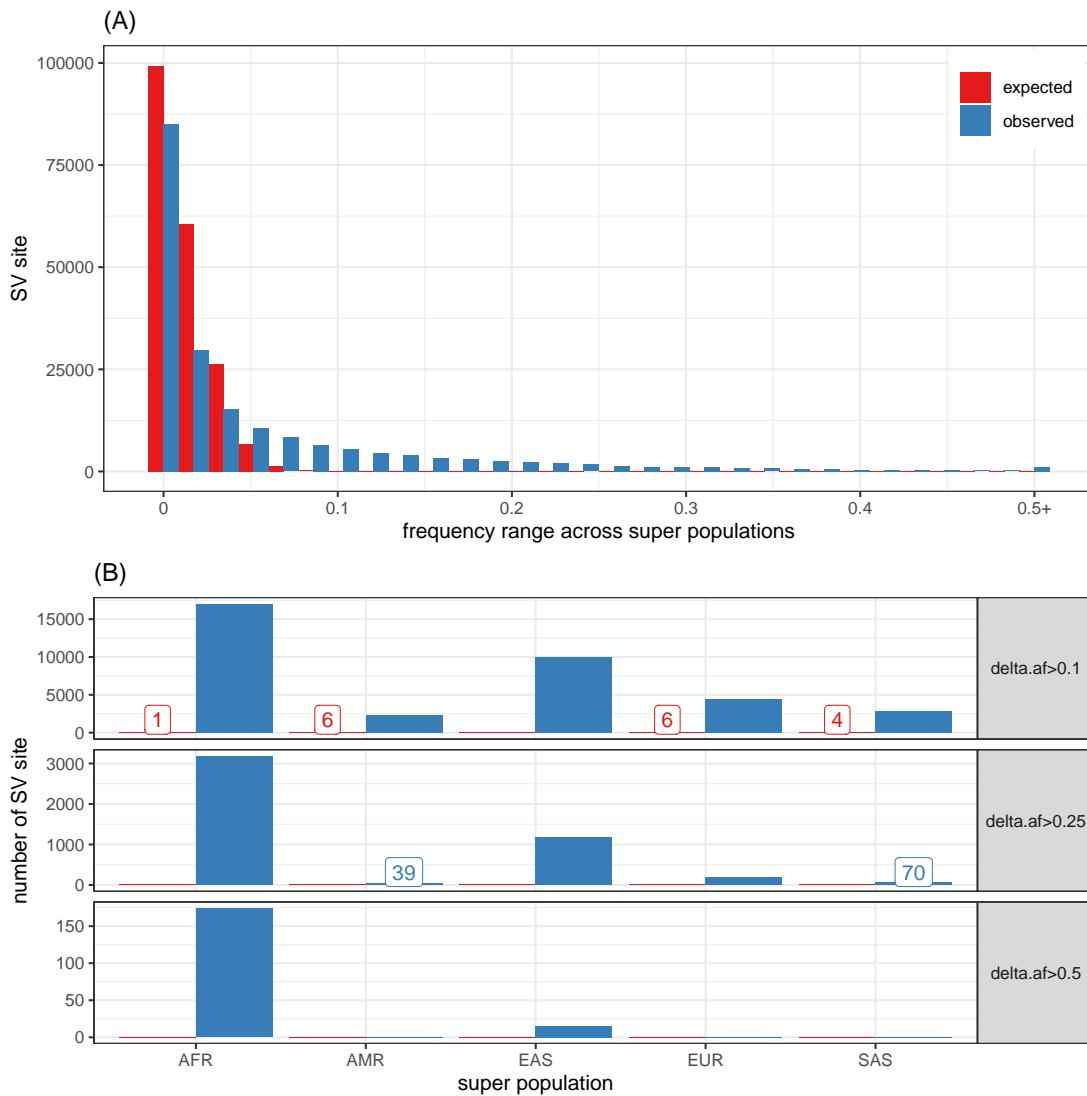
**Figure 4.32: SV sites with population specific frequency patterns in the MESA cohort.**  
The two thousand MESA samples were clustered in 6 clusters based on the top 3 principal components (A-B). (C) The range of frequency, i.e. the difference between the maximum and minimum frequencies, across the sample clusters was larger than expected (permutation) for thousands of SV sites. (D) Two clusters (x-axis) show high number of population-specific SV sites (y-axis), defined as deviating from the median allele frequency by at least 10%, 25%, or 50% (panels).

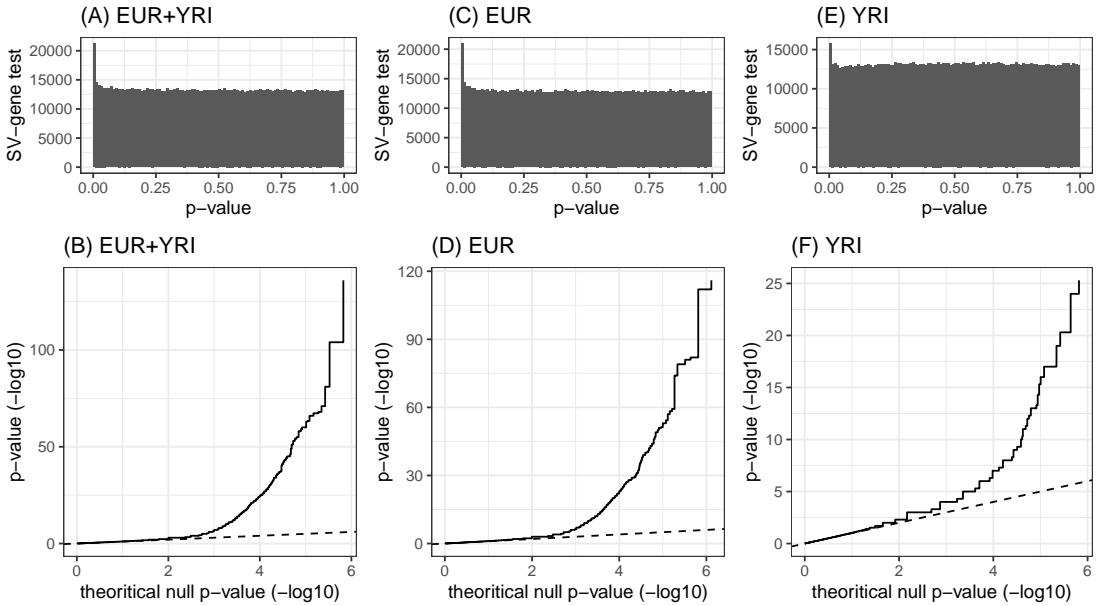


**Figure 4.33: Principal component analysis from structural variant genotypes across the 2,504 unrelated individuals in the 1000 Genomes Project dataset.** Principal component analysis was performed on the SV genotypes predicted by vg across the 166,199 SV loci. Individuals are colored according to their assigned “super population” in the 1000 Genomes Project. (A) shows the first and second principal components, (B) shows the third and fourth components.

Pipeline	Var Type	TP	FN	FP	Recall	Precision	F1
BWA-MEM	INDELS	522,705	2,764	2,214	0.994740	0.995955	0.995347
	SNPS	3,346,810	18,317	14,146	0.994557	0.995792	0.995174
DRAGEN	INDELS	522,672	2,797	<b>2,043</b>	0.994677	<b>0.996265</b>	0.995471
	SNPS	3,348,116	17,011	<b>10,799</b>	0.994945	<b>0.996786</b>	0.995865
VG-MAP	INDELS	522,905	2,564	2,260	0.995121	0.995873	0.995497
	SNPS	3,348,892	16,235	12,052	0.995176	0.996415	0.995795
Giraffe primary	INDELS	522,834	2,635	2,316	0.994985	0.995771	0.995378
	SNPS	3,350,496	14,631	11,308	0.995652	0.996637	0.996144
Giraffe	INDELS	<b>523,148</b>	<b>2,321</b>	2,177	<b>0.995583</b>	0.996026	<b>0.995804</b>
	SNPS	<b>3,351,672</b>	<b>13,455</b>	11,167	<b>0.996002</b>	0.996680	<b>0.996341</b>
fast Giraffe	INDELS	523,119	2,350	2,178	0.995528	0.996024	0.995776
	SNPS	3,351,207	13,920	11,503	0.995863	0.996580	0.996222

**Table 4.12: Hap.py performance of linear and graph-based pipelines against grch38-based references using 250bp paired reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.**

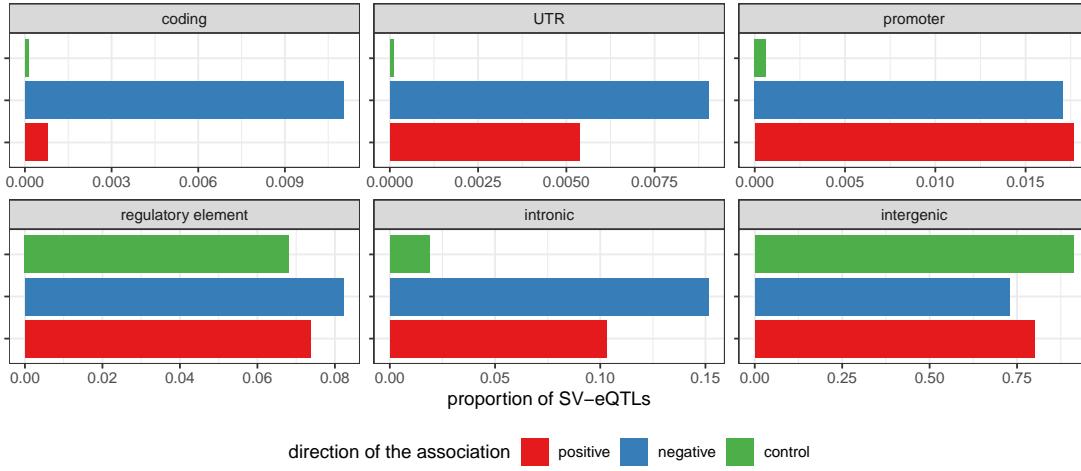




**Figure 4.35: P-value distribution of the SV vs gene expression tests in the GEUVADIS dataset.** Three analysis were performed: joint European and Yoruba populations (A-B), European populations only (C-D), and the Yoruba population (YRI) only (E-F). (A,C,E) show the p-value distribution, uniform except for the peak of low p-values. (B,D,F) show the QQ plots for the three analysis.

Pipeline	TP	FP	FN	Precision	Sensitivity	F-measure
BWA-MEM	3,370,702	1,678	4,248	0.9995	0.9987	0.9991
DRAGEN	3,371,710	<b>1,462</b>	3,296	<b>0.9996</b>	0.9990	0.9993
VG-MAP	3,372,845	3,121	2,141	0.9991	<b>0.9994</b>	0.9992
Giraffe primary	3,371,806	1,501	3,143	<b>0.9996</b>	0.9991	0.9993
Giraffe	<b>3,373,044</b>	1,543	<b>1,934</b>	0.9995	<b>0.9994</b>	<b>0.9995</b>
fast Giraffe	3,372,947	1,684	2,028	0.9995	<b>0.9994</b>	0.9994

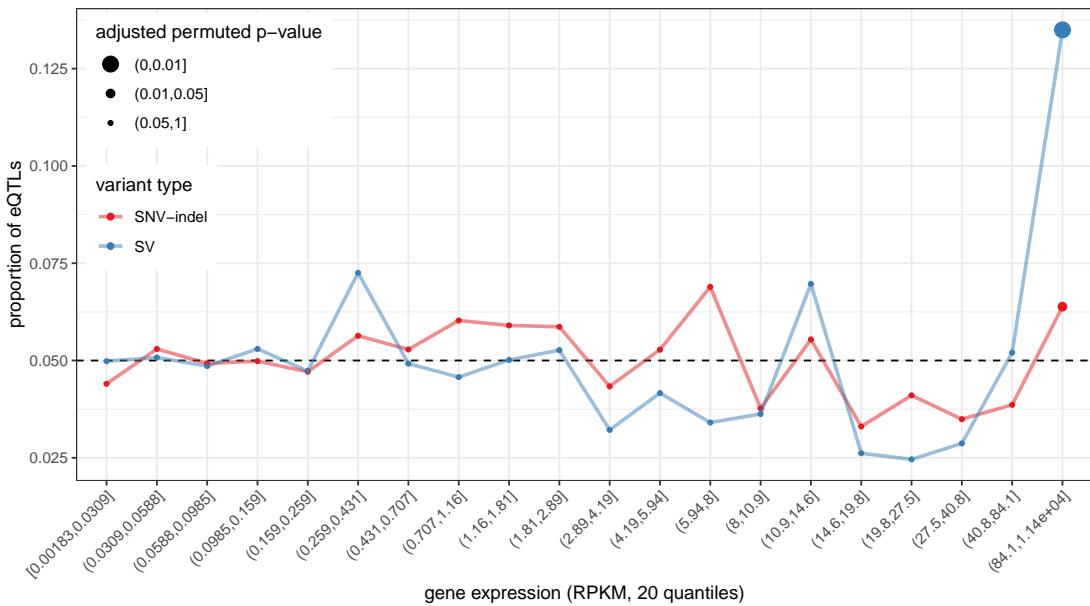
**Table 4.13:** VCFeval performance of linear and graph-based pipelines against grch38-based references using 150bp paired-end reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions only found in the 1000GP variant set used in constructing the graph references used by VG-MAP and Giraffe. Best values in each column are highlighted in bold text.



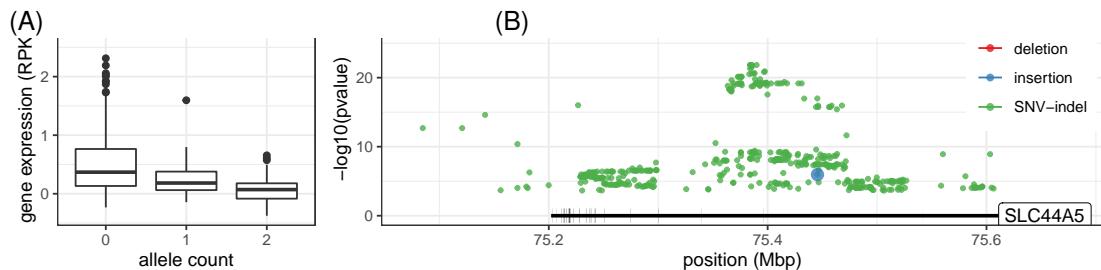
**Figure 4.36: Location of the SV-eQTLs around the associated gene** The x-axis reports the proportion of SV-eQTLs overlapping each annotation group (panels). A SV-eQTL is either positively (red) or negatively (blue) associated with gene expression, i.e. each additional SV allele correlates with increased or decreased gene expression respectively. The control bar (green) represents the expected distribution of common SVs around control genes. Control genes were chosen to match the size distribution of genes involved in eQTLs. *UTR: untranslated region.*

Pipeline	TP	FP	FN	Precision	Sensitivity	F-measure
BWA-MEM	331,019	20,738	19,972	0.9410	0.9429	0.9420
DRAGEN	331,381	<b>15,819</b>	19,643	<b>0.9544</b>	0.9438	0.9491
VG-MAP	331,322	17,144	19,694	0.9508	0.9437	0.9472
Giraffe primary	333,362	16,714	17,638	0.9523	0.9496	0.9509
Giraffe	<b>334,122</b>	16,144	<b>16,901</b>	0.9539	<b>0.9517</b>	<b>0.9528</b>
fast Giraffe	333,452	17,182	17,569	0.9510	0.9498	0.9504

**Table 4.14:** VCFeval performance of linear and graph-based pipelines against grch38-based references using 150bp paired-end reads with respect to HG002 GIAB v4.2.1 truth variant call sets in high confidence regions excluding the 1000GP variant set used in constructing the graph references used by VG-MAP and Giraffe. Best values in each column are highlighted in bold text.



**Figure 4.37: Enrichment of SV-eQTLs in highly expressed genes** Genes were split in 20 expression groups of equal size (quantiles). The graph shows the proportion of eQTLs (y-axis) in each expression quantile (x-axis). We see a strong enrichment for SV-eQTLs (blue) in the last quantile representing the most expressed genes. The enrichment is also significant (point size) if only weaker for SNV/indel eQTLs. The horizontal dotted line represents the expected value, i.e. 1/20 of the eQTLs in each quantile group.



**Figure 4.38: Example of SV lead-eQTL** Association between an intronic 5,405 bp deletion and the gene expression of the *SLC44A5* gene. (A) Each additional allele is associated with a decrease in gene expression. (B) shows the position of significant eQTLs (SNV/indels in green, deletion in red, insertion in blue). The y-axis represents the significance of the association, with the top eQTL being the highest point. The deletion is the most significant eQTLs and the significance decreases the farther the variant is from it.

Pipeline	TP	FP	FN	Precision	Sensitivity	F-measure
BWA-MEM + Dragen	3,807,536	23,443	24,345	0.9939	0.9936	0.9938
Giraffe + Dragen	<b>3,812,963</b>	18,143	<b>18,922</b>	0.9953	<b>0.9951</b>	0.9952
Dragen MAP + Dragen	3,808,794	18,738	23,081	0.9951	0.9940	0.9945
BWA-MEM + DeepVariant	3,808,209	<b>6,680</b>	24,173	<b>0.9982</b>	0.9937	0.9960
Giraffe + DeepVariant	3,812,716	6,889	19,898	<b>0.9982</b>	0.9948	<b>0.9965</b>

Table 4.15: RTG VCFeval performance between DeepVariant and Dragen variant callers on grch38-based linear and giraffe 1000GP mappers using 150bp paired-end reads with respect to HG003 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.

Pipeline	Var Type	TP	FN	FP	Recall	Precision	F1
BWA-MEM + Dragen	INDELS	501,574	2,927	2,789	0.994198	0.994696	0.994447
	SNPS	3,306,162	21,331	20,642	0.993589	0.993797	0.993693
Giraffe + Dragen	INDELS	<b>501,993</b>	<b>2,508</b>	2,566	<b>0.995029</b>	0.995122	0.995075
	SNPS	<b>3,311,179</b>	<b>16,317</b>	15,563	<b>0.995096</b>	0.995323	0.995210
Dragen MAP + Dragen	INDELS	501,770	2,731	2,644	0.994587	0.994972	0.994780
	SNPS	3,307,236	20,260	16,078	0.993911	0.995163	0.994537
BWA-MEM + DeepVariant	INDELS	501,433	3,068	<b>1,393</b>	0.993919	<b>0.997342</b>	0.995627
	SNPS	3,306,484	21,012	5,268	0.993685	0.998410	0.996042
Giraffe + DeepVariant	INDELS	501,919	2,582	1,851	0.994882	0.996476	<b>0.995678</b>
	SNPS	3,310,338	17,158	<b>5,015</b>	0.994844	<b>0.998488</b>	<b>0.996662</b>

Table 4.16: Hap.py performance between DeepVariant and Dragen variant callers on grch38-based linear and giraffe 1000GP mappers using 150bp paired-end reads with respect to HG003 GIAB v4.2.1 truth variant call sets in high confidence regions. Best values in each column are highlighted in bold text.

Pre-empted jobs	Dataset	Workflows	Core.hour
0	all	2910	194.388
	MESA	1221	197.981
	1000GP	1689	191.791
1	all	1358	220.046
	MESA	517	222.205
	1000GP	841	218.718
2	all	439	240.992
	MESA	172	248.246
	1000GP	267	236.318
3+	all	209	270.351
	MESA	88	282.007
	1000GP	121	261.874

Table 4.17: **Computing resources used when genotyping 2,000 samples from the MESA cohort and 3,202 samples from the 1000 Genomes Project dataset.** Pre-emptible instances on Google Cloud were used for all jobs. As expected, the average resource required was higher for workflows where some jobs were pre-empted. Note: The sequencing reads were down-sampled to  $\sim 20x$  coverage.

SV type	Number of non-reference alleles			Proportion of variants
	Offspring	Mother	Father	
Deletion	1	0	0	0.0255409
	0	2	0	0.0052011
	0	0	2	0.0049499
	2	0	0	0.0042598
	1	2	2	0.0021315
Insertion	1	0	0	0.0224647
	1	2	2	0.0045391
	0	2	0	0.0037815
	0	0	2	0.0036952
	2	0	0	0.0030867

Table 4.18: **Most frequent Mendelian error for SVs in the 1000GP trios.** The table shows the 5 most frequent Mendelian error configurations for deletions and insertions. For both SV types, the most frequent error is a predicted heterozygous variant in the offspring while both parents are predicted homozygous for the reference allele. Here all variants were included, i.e. no filtering based on genotype qualities was applied.

Public SV catalog	Dataset	Novel	Prop. of SVs in public catalog covered	Prop. of common SVs in public catalog covered
1000GP	MESA	0.932	0.160	0.844
	1000GP	0.932	0.160	0.844
gnomAD-SV	MESA	0.673	0.088	0.628
	1000GP	0.670	0.087	0.627

Table 4.19: **Comparing SVs genotyped in the two datasets (MESA: 2,000 samples; 1000GP: 2,504 samples) with two public SV catalogs.** The table shows the proportion of novel SVs (not present in the public SV catalogs), and the proportion of SVs in the public catalog that are present in our results, either considering all variants or common variants (frequency greater or equal to 5%). Deletions were matched if their reciprocal overlap was at least 30%, insertions if located at less than 200 bp from each other and their size at least 30% similar. Annotated simple repeats were used to extend the matching.

[supptab-S18-sv-eqtl.xlsx](#)

Table 4.20: **Gene families enriched in SV-eQTLs and eGenes with only SV-eQTLs.** Excel file containing two sheets. The first lists gene families enriched in SV-eQTLs. The gene families were defined using the HUGO Gene Nomenclature Committee (HGNC) resource. The second lists the SV-eQTLs in genes where there are only SV associations, i.e. no SNV/indel eQTLs.

Task	CPU	Memory (Gb)	Dataset	Core.hour
CRAM conversion	8	50	all	12.867 (7.6%)
CRAM conversion	8	50	MESA	13.22 (7.6%)
CRAM conversion	8	50	1000GP	12.636 (7.6%)
mapping	32	100	all	146.261 (75.2%)
mapping	32	100	MESA	157.66 (77.1%)
mapping	32	100	1000GP	138.782 (73.9%)
genotyping	16	100	all	24.891 (17.2%)
genotyping	16	100	MESA	27.112 (15.4%)
genotyping	16	100	1000GP	23.433 (18.5%)

Table 4.21: **Computing resources for each task in the WDL genotyping workflow.** The table shows numbers for the workflows where no jobs were pre-empted.

Mapper	Reads	Graph	vg version	Docker	xg	gcsa	lcp	gbwt	min	gg	dist
VG-MAP	150 bp	1000GP	v1.31.0	Quay	X	X	X				
VG-MAP	250 bp	1000GP	v1.31.0	Quay	X	X	X				
Giraffe	150 bp	Primary	v1.31.0	Quay	X			X	X	X	X
Giraffe	250 bp	Primary	v1.31.0	Quay	X			X	X	X	X
Giraffe	150 bp	1000GP	v1.31.0	Quay	X			X	X	X	X
Giraffe	250 bp	1000GP	v1.31.0	Quay	X			X	X	X	X
Fast Giraffe	150 bp	1000GP	v1.31.0	Quay	X			X	X	X	X
Fast Giraffe	250 bp	1000GP	v1.31.0	Quay	X			X	X	X	X

Table 4.22: Table of parameters for GRCh38-based genotyping experiment vg runs

# **Chapter 5**

## **Long read mapping**

### **5.1 Preface**

# **Chapter 6**

## **Discussion**

# **Appendix A**

## **Additional short read giraffe materials**

### **A.1 Preface**

This section contains materials from the supplement of short read giraffe paper [80] that are referenced in the main text but not related to my portion of the paper.

# Bibliography

- [1] Hidden Treasures - Warm Up – precisionFDA. <https://precision.fda.gov/challenges/1/view/results>, retrieved on 2020-11-11, currently unavailable.
- [2] Truth Challenge V2: Calling Variants from Short and Long Reads in Difficult-to-Map Regions – precisionFDA. <https://precision.fda.gov/challenges/10/view/results>, retrieved on 2020-11-11, currently unavailable.
- [3] Picard toolkit, 2019. <http://broadinstitute.github.io/picard/>.
- [4] Illumina/hap.py, November 2020. <https://github.com/Illumina/hap.py>.
- [5] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.
- [6] Alexej Abyzov, Alexander E Urban, Michael Snyder, and Mark Gerstein. Cnvator: an approach to discover, genotype, and characterize typical and atypical cnvs from family and population genome sequencing. *Genome research*, 21(6):974–984, 2011.
- [7] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 349, New York, New York, USA, 2013. ACM Press.
- [8] 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, Wesley C. Warren, Vincent Magrini, Sean D. McGrath, Yang I. Li, Richard K. Wilson, and Evan E. Eichler. Characterizing the Major Structural Variant Alleles of the Human Genome. *Cell*, 176(3):663–675.e19, January 2019.
- [9] Sara Ballouz, Alexander Dobin, and Jesse A. Gillis. Is it time to change the reference genome? *Genome Biology*, 20(1):159, August 2019.
- [10] Eduardo E. Benarroch. Anoctamins (TMEM16 proteins): Functions and involvement in neurologic disease. *Neurology*, 89(7):722–729, August 2017.
- [11] Wolfgang Beyer, Adam M Novak, Glenn Hickey, Jeffrey Chan, Vanessa Tan, Benedict Paten, and Daniel R Zerbino. Sequence tube maps: making graph genomes intuitive to commuters. *Bioinformatics*, 35(24):5318, 2019.

- [12] Diane E Bild, David A Bluemke, Gregory L Burke, Robert Detrano, Ana V Diez Roux, Aaron R Folsom, Philip Greenland, David R JacobsJr, Richard Kronmal, Kiang Liu, et al. Multi-ethnic study of atherosclerosis: objectives and design. *American journal of epidemiology*, 156(9):871–881, 2002.
- [13] David A Bluemke, Richard A Kronmal, João AC Lima, Kiang Liu, Jean Olson, Gregory L Burke, and Aaron R Folsom. The relationship of left ventricular mass and geometry to incident cardiovascular events: the mesa (multi-ethnic study of atherosclerosis) study. *Journal of the American College of Cardiology*, 52(25):2148–2155, 2008.
- [14] 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, Susan Fairley, Alexi Runnels, Lara Winterkorn, Ernesto Lowy-Gallego, , Paul Flicek, Soren Germer, Harrison Brand, Ira M. Hall, Michael E. Talkowski, Giuseppe Narzisi, and Michael C. Zody. High coverage whole genome sequencing of the expanded 1000 genomes project cohort including 602 trios. *bioRxiv*, 2021.
- [15] Mark J. P. Chaisson, Ashley D. Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J. Gardner, Oscar L. Rodriguez, Li Guo, Ryan L. Collins, Xian Fan, Jia Wen, Robert E. Handsaker, Susan Fairley, Zev N. Kronenberg, Xiangmeng Kong, Fereydoun Hormozdiari, Dillon Lee, Aaron M. Wenger, Alex R. Hastie, Danny Antaki, Thomas Anantharaman, Peter A. Audano, Harrison Brand, Stuart Cantsilieris, Han Cao, Eliza Cerveira, Chong Chen, Xintong Chen, Chen-Shan Chin, Zechen Chong, Nelson T. Chuang, Christine C. Lambert, Deanna M. Church, Laura Clarke, Andrew Farrell, Joey Flores, Timur Galeev, David U. Gorkin, Madhusudan Gujral, Victor Guryev, William Haynes Heaton, Jonas Korlach, Sushant Kumar, Jee Young Kwon, Ernest T. Lam, Jong Eun Lee, Joyce Lee, Wan-Ping Lee, Sau Peng Lee, Shantao Li, Patrick Marks, Karine Viaud-Martinez, Sascha Meiers, Katherine M. Munson, Fabio C. P. Navarro, Bradley J. Nelson, Conor Nodzak, Amina Noor, Sofia Kyriazopoulou-Panagiotopoulou, Andy W. C. Pang, Yunjiang Qiu, Gabriel Rosanio, Mallory Ryan, Adrian Stütz, Diana C. J. Spierings, Alistair Ward, AnneMarie E. Welch, Ming Xiao, Wei Xu, Chengsheng Zhang, Qihui Zhu, Xiangqun Zheng-Bradley, Ernesto Lowy, Sergei Yakneen, Steven McCarroll, Goo Jun, Li Ding, Chong Lek Koh, Bing Ren, Paul Flicek, Ken Chen, Mark B. Gerstein, Pui-Yan Kwok, Peter M. Lansdorp, Gabor T. Marth, Jonathan Sebat, Xinghua Shi, Ali Bashir, Kai Ye, Scott E. Devine, Michael E. Talkowski, Ryan E. Mills, Tobias Marschall, Jan O. Korbel, Evan E. Eichler, and Charles Lee. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature Communications*, 10(1), December 2019.
- [16] 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, July 2020.
- [17] Sai Chen, Peter Krusche, Egor Dolzhenko, Rachel M. Sherman, Roman Petrovski, Felix Schlesinger, Melanie Kirsche, David R. Bentley, Michael C. Schatz, Fritz J. Sedlazeck,

- and Michael A. Eberle. Paragraph: a graph-based structural variant genotyper for short-read sequence data. *Genome Biology*, 20(1):291, December 2019.
- [18] Colby Chiang, Alexandra J Scott, Joe R Davis, Emily K Tsang, Xin Li, Yungil Kim, Tarik Hadzic, Farhan N Damani, Liron Ganel, GTEx Consortium, Stephen B Montgomery, Alexis Battle, Donald F Conrad, and Ira M Hall. The impact of structural variation on human gene expression. *Nature Genetics*, 49(5):692–699, May 2017.
  - [19] Chen-Shan Chin, Justin Wagner, Qiandong Zeng, Erik Garrison, Shilpa Garg, Arkarachai Fungtammasan, Mikko Rautiainen, Sergey Aganezov, Melanie Kirsche, Samantha Zarate, et al. A diploid assembly-based benchmark for variants in the major histocompatibility complex. *Nature Communications*, 11(1):1–9, 2020.
  - [20] Deanna M. Church, Valerie A. Schneider, Tina Graves, Katherine Auger, Fiona Cunningham, Nathan Bouk, Hsiu-Chuan Chen, Richa Agarwala, William M. McLaren, Graham R. S. Ritchie, Derek Albracht, Milinn Kremitzki, Susan Rock, Holland Kotkiewicz, Colin Kremitzki, Aye Wollam, Lee Trani, Lucinda Fulton, Robert Fulton, Lucy Matthews, Siobhan Whitehead, Will Chow, James Torrance, Matthew Dunn, Glenn Harden, Glen Threadgold, Jonathan Wood, Joanna Collins, Paul Heath, Guy Griffiths, Sarah Pelan, Darren Grafham, Evan E. Eichler, George Weinstock, Elaine R. Mardis, Richard K. Wilson, Kerstin Howe, Paul Flicek, and Tim Hubbard. Modernizing Reference Genome Assemblies. *PLOS Biology*, 9(7):e1001091, July 2011.
  - [21] John G Cleary, Ross Braithwaite, Kurt Gaastra, Brian S Hilbush, Stuart Inglis, Sean A Irvine, Alan Jackson, Richard Littin, Mehul Rathod, David Ware, et al. Comparing variant call files for performance benchmarking of next-generation sequencing variant calling pipelines. *bioRxiv*, 2015.
  - [22] The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, page bbw089, October 2016.
  - [23] Danang Crysnto and Hubert Pausch. Bovine breed-specific augmented reference graphs facilitate accurate sequence read mapping and unbiased variant discovery. *Genome Biology*, 21(1):184, July 2020.
  - [24] Vachik S. Dave and Mohammad Al Hasan. TopCom: Index for Shortest Distance Query in Directed Graph. In Qiming Chen, Abdelkader Hameurlain, Farouk Toumani, Roland Wagner, and Hendrik Decker, editors, *Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 471–480, Cham, 2015. Springer International Publishing.
  - [25] 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, January 2018.

- [26] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Matematik*, 1(1):269–271, December 1959.
- [27] Hristo N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Fabrizio d’Amore, Paolo Giulio Franciosa, and Alberto Marchetti-Spaccamela, editors, *Graph-Theoretic Concepts in Computer Science*, volume 1197, pages 151–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
- [28] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.
- [29] Peter Ebert, Peter A. Audano, Qihui Zhu, Bernardo Rodriguez-Martin, David Porubsky, Marc Jan Bonder, Arvis Sulovari, Jana Ebler, Weichen Zhou, Rebecca Serra Mari, Feyza Yilmaz, Xuefang Zhao, PingHsun Hsieh, Joyce Lee, Sushant Kumar, Jiadong Lin, Tobias Rausch, Yu Chen, Jingwen Ren, Martin Santamarina, Wolfram Höps, Hufsah Ashraf, Nelson T. Chuang, Xiaofei Yang, Katherine M. Munson, Alexandra P. Lewis, Susan Fairley, Luke J. Tallon, Wayne E. Clarke, Anna O. Basile, Marta Byrska-Bishop, André Corvelo, Uday S. Evani, Tsung-Yu Lu, Mark J.P. Chaisson, Junjie Chen, Chong Li, Harrison Brand, Aaron M. Wenger, Maryam Gharegiani, William T. Harvey, Benjamin Raeder, Patrick Hasenfeld, Allison A. Regier, Haley J. Abel, Ira M. Hall, Paul Flicek, Oliver Stegle, Mark B. Gerstein, Jose M.C. Tubio, Zepeng Mu, Yang I. Li, Xinghua Shi, Alex R. Hastie, Kai Ye, Zechen Chong, Ashley D. Sanders, Michael C. Zody, Michael E. Talkowski, Ryan E. Mills, Scott E. Devine, Charles Lee, Jan O. Korbel, Tobias Marschall, and Evan E. Eichler. Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science*, February 2021.
- [30] Jana Ebler, Alexander Schönhuth, and Tobias Marschall. Genotyping inversions and tandem duplications. *Bioinformatics*, 33(24):4015–4023, December 2017.
- [31] Hannes P. Eggertsson, Snaedis Kristmundsdottir, Doruk Beyter, Hakon Jonsson, Astros Skuladottir, Marteinn T. Hardarson, Daniel F. Gudbjartsson, Kari Stefansson, Bjarni V. Halldorsson, and Pall Melsted. GraphTyper2 enables population-scale genotyping of structural variation using pangenome graphs. *Nature Communications*, 10(1):5402, December 2019.
- [32] 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*, 2020.
- [33] Jordan M. Eizenga, Adam M. Novak, Jonas A. Sibbesen, Simon Heumos, Ali Ghaffaari, Glenn Hickey, Xian Chang, Josiah D. Seaman, Robin Rounthwaite, Jana Ebler, Mikko Rautiainen, Shilpa Garg, Benedict Paten, Tobias Marschall, Jouni Sirén, and Erik Garrison. Pangenome Graphs. *Annual Review of Genomics and Human Genetics*, 21(1):139–162, August 2020.

- [34] Brent Ewing and Phil Green. Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome research*, 8(3):186–194, 1998.
- [35] Maud Fagny, Joseph N. Paulson, Marieke L. Kuijjer, Abhijeet R. Sonawane, Cho-Yi Chen, Camila M. Lopes-Ramos, Kimberly Glass, John Quackenbush, and John Platig. Exploring regulation in tissues with eQTL networks. *Proceedings of the National Academy of Sciences*, 114(37):E7841–E7850, September 2017.
- [36] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.
- [37] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, October 2018.
- [38] Genome Aggregation Database Production Team, Genome Aggregation Database Consortium, Ryan L. Collins, Harrison Brand, Konrad J. Karczewski, Xuefang Zhao, Jessica Alföldi, Laurent C. Francioli, Amit V. Khera, Chelsea Lowther, Laura D. Gauthier, Harold Wang, Nicholas A. Watts, Matthew Solomonson, Anne O’Donnell-Luria, Alexander Baumann, Ruchi Munshi, Mark Walker, Christopher W. Whelan, Yongqing Huang, Ted Brookings, Ted Sharpe, Matthew R. Stone, Elise Valkanas, Jack Fu, Grace Tiao, Kristen M. Laricchia, Valentin Ruano-Rubio, Christine Stevens, Namrata Gupta, Caroline Cusick, Lauren Margolin, Kent D. Taylor, Henry J. Lin, Stephen S. Rich, Wendy S. Post, Yii-Der Ida Chen, Jerome I. Rotter, Chad Nusbaum, Anthony Philippakis, Eric Landier, Stacey Gabriel, Benjamin M. Neale, Sekar Kathiresan, Mark J. Daly, Eric Banks, Daniel G. MacArthur, and Michael E. Talkowski. A structural variation reference for medical and population genetics. *Nature*, 581(7809):444–451, May 2020.
- [39] Ali Ghaffaari and Tobias Marschall. Fully-sensitive seed finding in sequence graphs using a hybrid index. *Bioinformatics*, 35(14):i81–i89, 2019.
- [40] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.
- [41] 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):35, December 2020.
- [42] Illumina. Accuracy Improvements in Germline Small Variant Calling with the DRAGEN Platform. <https://science-docs.illumina.com/documents/Informatics/dragen-v3-accuracy-appnote-html-970-2019-006/Content/Source/Informatics/Dragen/dragen-v3-accuracy-appnote-970-2019-006/dragen-v3-accuracy-appnote-970-2019-006.html>.

- [43] International HapMap Consortium et al. Integrating ethics and science in the international hapmap project. *Nature reviews. Genetics*, 5(6):467, 2004.
- [44] International HapMap Consortium et al. A haplotype map of the human genome. *Nature*, 437(7063):1299, 2005.
- [45] Chirag Jain, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Validating Paired-End Read Alignments in Sequence Graphs. In Katharina T. Huber and Dan Gusfield, editors, *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*, volume 143 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [46] 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 Sciences*, 87(6):2264–2268, 1990.
- [47] 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, August 2019.
- [48] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, May 2017.
- [49] Peter Krusche, Len Trigg, Paul C Boutros, Christopher E Mason, M Francisco, Benjamin L Moore, Mar Gonzalez-Porta, Michael A Eberle, Zivana Tezak, Samir Lababidi, et al. Best practices for benchmarking germline small-variant calls in human genomes. *Nature biotechnology*, 37(5):555–560, 2019.
- [50] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, March 2012.
- [51] Ulrich Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Raubal, M., Sliwinski, A., Kuhn, W. (eds.) *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, 22:12, 2004.
- [52] 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.
- [53] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, May 2013.
- [54] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, July 2016.

- [55] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018.
- [56] Heng Li, Xiaowen Feng, and Chong Chu. The design and construction of reference pangenome graphs with minigraph. *Genome Biology*, 21(1):265, 2020.
- [57] Medhat Mahmoud, Nastassia Gobet, Diana Ivette Cruz-Dávalos, Ninon Mounier, Christophe Dessimoz, and Fritz J. Sedlazeck. Structural variant calling: the long and the short of it. *Genome Biology*, 20(1):1–14, December 2019.
- [58] Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning Graphs to Speed Up Dijkstra’s Algorithm. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and Sotiris E. Nikoletseas, editors, *Experimental and Efficient Algorithms*, volume 3503, pages 189–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
- [59] Jean Monlong. [github.com/jmonlong/wdl-workflows/bcftools\\_merge](https://github.com/jmonlong/wdl-workflows/bcftools_merge), November 2020. <https://doi.org/10.5281/zenodo.4290655>.
- [60] Jean Monlong. [github.com/vgteam/vg\\_wdl/vg\\_mapgaffe\\_call\\_sv\\_cram](https://github.com/vgteam/vg_wdl/vg_mapgaffe_call_sv_cram), November 2020.
- [61] Lisle E Mose, Charles M Perou, and Joel S Parker. Improved indel detection in DNA and RNA via realignment with ABRA2. *Bioinformatics*, 35(17):2966–2973, 01 2019.
- [62] National Heart, Lung, and Blood Institute, National Institutes of Health, U.S. Department of Health and Human Services. The NHLBI BioData Catalyst. August 2020.
- [63] Benedict Paten, Jordan M Eizenga, Yohei M Rosen, Adam M Novak, Erik Garrison, and Glenn Hickey. Superbubbles, Ultrabubbles, and Cacti. *Journal of Computational Biology*, 25:15, 2018.
- [64] 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, May 2017.
- [65] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T. Afshar, Sam S. Gross, Lizzie Dorfman, Cory Y. McLean, and Mark A. DePristo. A universal SNP and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36(10):983–987, November 2018.
- [66] Jacob Pritt, Nae-Chyun Chen, and Ben Langmead. FORGe: prioritizing variants for graph genomes. *Genome Biology*, 19:220, 2018.
- [67] Andrey Prjibelski, Dmitry Antipov, Dmitry Meleshko, Alla Lapidus, and Anton Korobeynikov. Using SPAdes De Novo Assembler. *Current Protocols in Bioinformatics*, 70(1), 2020.

- [68] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme. In *2012 IEEE 28th International Conference on Data Engineering*, pages 462–473, April 2012. ISSN: 2375-026X, 1063-6382, 1063-6382.
- [69] Goran Rakocevic. Fast and accurate genomic analyses using genome graphs. *Nature Genetics*, 51:14, 2019.
- [70] Goran Rakocevic, Vladimir Semenyuk, Wan-Ping Lee, James Spencer, John Browning, Ivan J. Johnson, Vladan Arsenijevic, Jelena Nadj, Kaushik Ghose, Maria C. Suciu, Sun-Gou Ji, Gülfem Demir, Lizao Li, Berke Ç Toptaş, Alexey Dolgoborodov, Björn Pollex, Iosif Spulber, Irina Glotova, Péter Kómár, Andrew L. Stachyra, Yilong Li, Milos Popovic, Morten Källberg, Amit Jain, and Deniz Kural. Fast and accurate genomic analyses using genome graphs. *Nature Genetics*, 51(2):354–362, February 2019.
- [71] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19):3599–3607, October 2019.
- [72] Mikko Rautiainen and Tobias Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21(1):253, September 2020.
- [73] 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.
- [74] Melanie Schirmer, Rosalinda D’Amore, Umer Z Ijaz, Neil Hall, and Christopher Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC bioinformatics*, 17(1):1–15, 2016.
- [75] 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):R98, September 2009.
- [76] A. A. Shabalin. Matrix eQTL: ultra fast eQTL analysis via large matrix operations. *Bioinformatics*, 28(10):1353–1358, May 2012.
- [77] Rachel M. Sherman and Steven L. Salzberg. Pan-genomics in the human genome era. *Nature Reviews Genetics*, 21(4):243–254, April 2020.
- [78] 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. <https://doi.org/10.1101/2021.03.26.437240>.
- [79] Jouni Sirén, Erik Garrison, Adam M. Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, 2020.

- [80] 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. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science*, 374(6574):abg8871, December 2021.
- [81] Jouni Sirén, Jean Monlong, 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. Software and products for "Pangenomics enables genotyping known structural variants in 5,202 diverse genomes", 2021. <https://doi.org/10.5281/zenodo.4774363>.
- [82] Cricket A. Sloan, Esther T. Chan, Jean M. Davidson, Venkat S. Malladi, J. Seth Stratton, Benjamin C. Hitz, Idan Gabdank, Aditi K. Narayanan, Marcus Ho, Brian T. Lee, Laurence D. Rowe, Timothy R. Dreszer, Greg Roe, Nikhil R. Podduturi, Forrest Tanaka, Eurie L. Hong, and J. Michael Cherry. ENCODE data at the ENCODE portal. *Nucleic Acids Research*, 44(D1):D726–D732, January 2016.
- [83] P. H. Sudmant, S. Mallick, B. J. Nelson, F. Hormozdiari, N. Krumm, J. Huddleston, B. P. Coe, C. Baker, S. Nordenfelt, M. Bamshad, L. B. Jorde, O. L. Posukh, H. Sahakyan, W. S. Watkins, L. Yepiskoposyan, M. S. Abdullah, C. M. Bravi, C. Capelli, T. Hervig, J. T. S. Wee, C. Tyler-Smith, G. van Driem, I. G. Romero, A. R. Jha, S. Karachanak-Yankova, D. Toncheva, D. Comas, B. Henn, T. Kivisild, A. Ruiz-Linares, A. Sajantila, E. Metspalu, J. Parik, R. Villems, E. B. Starikovskaya, G. Ayodo, C. M. Beall, A. Di Rienzo, M. Hammer, R. Khusainova, E. Khusnutdinova, W. Klitz, C. Winkler, D. Labuda, M. Metspalu, S. A. Tishkoff, S. Dryomov, R. Sukernik, N. Patterson, D. Reich, and E. E. Eichler. Global diversity, population stratification, and selection of human copy number variation. *Science*, pages 1–16, August 2015.
- [84] 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, Miriam K. Konkel, Ankit Malhotra, Adrian M. Stütz, Xinghua Shi, Francesco Paolo Casale, Jieming Chen, Fereydoun Hormozdiari, Gargi Dayama, Ken Chen, Maika Malig, Mark J. P. Chaisson, Klaudia Walter, Sascha Meiers, Seva Kashin, Erik Garrison, Adam Auton, Hugo Y. K. Lam, Xinmeng Jasmine Mu, Can Alkan, Danny Antaki, Taejeong Bae, Eliza Cerveira, Peter Chines, Zechen Chong, Laura Clarke, Elif Dal, Li Ding, Sarah Emery, Xian Fan, Madhusudan Gujral, Fatma Kahveci, Jeffrey M. Kidd, Yu Kong, Eric-Wubbo Lameijer, Shane McCarthy, Paul Flicek, Richard a. Gibbs, Gabor Marth, Christopher E. Mason, Androniki Menelaou, Donna M. Muzny, Bradley J. Nelson, Amina Noor, Nicholas F. Parish, Matthew Pendleton, Andrew Quittadamo, Benjamin Raeder, Eric E. Schadt, Mallory Romanovitch, Andreas Schlattl, Robert Sebra, Andrey a. Shabalin, Andreas Untergasser, Jerilyn A. Walker, Min Wang, Fuli Yu, Chengsheng Zhang, Jing Zhang, Xiangqun Zheng-Bradley, Wanding Zhou, Thomas Zichner, Jonathan Sebat, Mark A. Batzer, Steven A.

- McCarroll, Ryan E. Mills, Mark B. Gerstein, Ali Bashir, Oliver Stegle, Scott E. Devine, Charles Lee, Evan E. Eichler, and Jan O. Korbel. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, September 2015.
- [85] O. Tange. Gnu parallel - the command-line power tool. *The USENIX Magazine*, 36(1):42–47, Feb 2011.
- [86] The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, September 2012.
- [87] The Geuvadis Consortium, Tuuli Lappalainen, Michael Sammeth, Marc R. Friedländer, Peter A. C. 'T Hoen, Jean Monlong, Manuel A. Rivas, Mar González-Porta, Natalja Kurbatova, Thasso Griebel, Pedro G. Ferreira, Matthias Barann, Thomas Wieland, Liliana Greger, Maarten Van Iterson, Jonas Almlöf, Paolo Ribeca, Irina Pulyakhina, Daniela Esser, Thomas Giger, Andrew Tikhonov, Marc Sultan, Gabrielle Bertier, Daniel G. MacArthur, Monkol Lek, Esther Lizano, Henk P. J. Buermans, Ismael Padioleau, Thomas Schwarzmayr, Olof Karlberg, Halit Ongen, Helena Kilpinen, Sergi Beltran, Marta Gut, Katja Kahlem, Vyacheslav Amstislavskiy, Oliver Stegle, Matti Pirinen, Stephen B. Montgomery, Peter Donnelly, Mark I. McCarthy, Paul Flicek, Tim M. Strom, Hans Lehrach, Stefan Schreiber, Ralf Sudbrak, Ángel Carracedo, Stylianos E. Antonarakis, Robert Häslar, Ann-Christine Syvänen, Gert-Jan Van Ommen, Alvis Brazma, Thomas Meitinger, Philip Rosenstiel, Roderic Guigó, Ivo G. Gut, Xavier Estivill, and Emmanouil T. Dermitzakis. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501(7468):506–511, September 2013.
- [88] Kavya Vaddadi, Rajgopal Srinivasan, and Naveen Sivadasan. Read Mapping on Genome Variation Graphs. In *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*, 2019.
- [89] Justin Wagner, Nathan D Olson, Lindsay Harris, Ziad Khan, Jesse Farek, Medhat Mahmoud, Ana Stankovic, Vladimir Kovacevic, Aaron M Wenger, William J Rowell, Chunlin Xiao, Byunggil Yoo, Neil Miller, Jeffrey A Rosenfeld, Bohan Ni, Samantha Zarate, Melanie Kirsche, Sergey Aganezov, Michael Schatz, Giuseppe Narzisi, Marta Byrska-Bishop, Wayne Clarke, Uday S Evani, Charles Markello, Kishwar Shafin, Xin Zhou, Arend Sidow, Vikas Bansal, Peter Ebert, Tobias Marschall, Peter Lansdorp, Vincent Hanlon, Carl-Adam Mattsson, Alvaro Martinez Barrio, Ian T Fiddes, Arkarachai Fungtammasan, Chen-Shan Chin, Fritz J Sedlazeck, Andrew Carroll, Marc Salit, Justin M Zook, and . Benchmarking challenging small variants with linked and long reads. *bioRxiv*, 2020.
- [90] Ryan R. Wick, Mark B. Schultz, Justin Zobel, and Kathryn E. Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20):3350–3352, October 2015.
- [91] Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth. SSW library: An SIMD Smith-Waterman C/C++ library for use in genomic applications. *PLOS ONE*, 8(12):e82138, 2013.

- [92] Justin M. Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E. Mason, Noah Alexander, Elizabeth Henaff, Alexa B.R. McIntyre, Dhruva Chandramohan, Feng Chen, Erich Jaeger, Ali Moshrefi, Khoa Pham, William Stedman, Tiffany Liang, Michael Saghbini, Zeljko Dzakula, Alex Hastie, Han Cao, Gintaras Deikus, Eric Schadt, Robert Sebra, Ali Bashir, Rebecca M. Truty, Christopher C. Chang, Natali Gulbahce, Keyan Zhao, Srinka Ghosh, Fiona Hyland, Yutao Fu, Mark Chaisson, Chunlin Xiao, Jonathan Trow, Stephen T. Sherry, Alexander W. Zaranek, Madeleine Ball, Jason Bobe, Preston Estep, George M. Church, Patrick Marks, Sofia Kyriazopoulou-Panagiotopoulou, Grace X.Y. Zheng, Michael Schnall-Levin, Heather S. Ordóñez, Patrice A. Mudivarti, Kristina Giorda, Ying Sheng, Karoline Bjarnesdatter Rypdal, and Marc Salit. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3(1):160025, December 2016.
- [93] Justin M. Zook, Brad Chapman, Jason Wang, David Mittelman, Oliver Hofmann, Winston Hide, and Marc Salit. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nature Biotechnology*, 32(3):246–251, March 2014.
- [94] Justin M. Zook, Nancy F. Hansen, Nathan D. Olson, Lesley Chapman, James C. Mullikin, Chunlin Xiao, Stephen Sherry, Sergey Koren, Adam M. Phillippy, Paul C. Boutros, Sayed Mohammad E. Sahraeian, Vincent Huang, Alexandre Rouette, Noah Alexander, Christopher E. Mason, Iman Hajirasouliha, Camir Ricketts, Joyce Lee, Rick Tearle, Ian T. Fiddes, Alvaro Martinez Barrio, Jeremiah Wala, Andrew Carroll, Noushin Ghaffari, Oscar L. Rodriguez, Ali Bashir, Shaun Jackman, John J. Farrell, Aaron M. Wenger, Can Alkan, Arda Soylev, Michael C. Schatz, Shilpa Garg, George Church, Tobias Marschall, Ken Chen, Xian Fan, Adam C. English, Jeffrey A. Rosenfeld, Weichen Zhou, Ryan E. Mills, Jay M. Sage, Jennifer R. Davis, Michael D. Kaiser, John S. Oliver, Anthony P. Catalano, Mark J. P. Chaisson, Noah Spies, Fritz J. Sedlazeck, and Marc Salit. A robust benchmark for detection of germline large deletions and insertions. *Nature Biotechnology*, 38(11):1347–1355, November 2020.