UNIVERSITY OF CALIFORNIA
SANTA CRUZ

USING LONG READS TO IMPROVE HAPLOTYPE PHASING, GENOME ASSEMBLY, AND GENE ANNOTATION

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

DOCTOR OF PHILOSOPHY

in

BIOMOLECULAR ENGINEERING AND BIOINFORMATICS

by

Marina Haukness

December 2022

The Dissertation of Marina Haukness
is approved:

Professor Benedict Paten, Chair

Professor David Haussler

Professor Richard Green

Peter F. Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures iv

List of Tables xii

Abstract xvi

Dedication xvii

Acknowledgments xviii

1 Introduction 1

2 Haplotype Phasing 13

3 Genome Assembly 63

4 Gene Annotation 107

5 Gene annotation on a fully complete, telomere-to-telomere human assembly 126

6 Gene annotation on a human pangenome 133

Bibliography 158

A List of publications 177

B MarginPhase appendix 180

C Shasta methods 195

D Shasta appendix 267

List of Figures

2.1 **An example of genotypes vs. haplotypes.** Genotypes describe which base are at a location in the genome. The genotypes at each of the three loci here are A/T, G/A, and T/C. Haplotypes designate which are inherited together on the same chromosome. In this example, the A/A/C variants are on the same chromosome and therefore the same haplotype, and the T/G/T variants are on the other. 14

2.2 **An example of clinically significant differences in haplotypes.** Two different possibilities for organization of the same genotype. In (a), the wild type promoter and gene are on the same chromosome and are therefore able to produce a normal amount of functional protein. In (b), only a reduced amount of functional protein is produced. 15

2.3 **Motivation and overview of diplotyping**. Gray sequences illustrate the haplotypes; the reads are shown in red and blue. The red reads originate from the upper haplotype, the blue ones from the lower. Genotyping each SNV individually would lead to the conclusion that all of them are heterozygous. Using the haplotype context reveals uncertainty about the genotype of the second SNV. 16

2.4 **Motivation and overview of diplotyping.** a Gray sequences illustrate the haplotypes; the reads are shown in red and blue. The red reads originate from the upper haplotype, the blue ones from the lower. Genotyping each SNV individually would lead to the conclusion that all of them are heterozygous. Using the haplotype context reveals uncertainty about the genotype of the second SNV. b Clockwise starting top left: first, sequencing reads aligned to a reference genome are given as input; second, the read alignments are used to nominate candidate variants (red vertical bars), which are characterized by the differences to the reference genome; third, a hidden Markov model (HMM) is constructed where each candidate variant gives rise to one "row" of states, representing possible ways of assigning each read to one of the two haplotypes as well as possible genotypes (see the "Methods" section for details); forth, the HMM is used to perform diplotyping, i.e., we infer genotypes of each candidate variant as well as how the alleles are assigned to haplotypes 26

2.5 **Reach of short read and long read technologies.** The callable and mappable regions for NA12878 spanning various repetitive or duplicated sequences on GRCh38 are shown. Feature locations are determined based on BED tracks downloaded from the UCSC Genome Browser [48]. Other than the Gencode regions [49, 50], all features are subsets of the Repeat Masker [51] track. Four coverage statistics for long reads (shades of red) and three for short reads (shades of blue) are shown. The labels "PacBio Mappable" and "Nanopore Mappable" describe areas where at least one primary read with GQ ≥ 30 has mapped, and "Long Read Mappable" describes where this is true for at least one of the long read technologies. "Long Read Callable" describes areas where both read technologies have coverage of at least 20 and less than twice the median coverage. "GIAB High Confidence," "GATK Callable," and "Short Read Mappable" are the regions associated with the evaluation callsets. For the feature-specific plots, the numbers on the right detail coverage over the feature and coverage over the whole genome (parenthesized) 32

2.6 **Precision and recall of MarginPhase on Nanopore and WhatsHap on PacBio datasets in GIAB high confidence regions.**Genotype concordance (bottom) (wrt. GIAB high confidence calls) of MarginPhase (mp, top) on Nanopore and WhatsHap (wh, middle) on PacBio (PB). Furthermore, genotype concordance for the intersection of the calls made by WhatsHap on the PacBio and MarginPhase on the Nanopore reads is shown (bottom) 35

2.7 **Genotyping errors (with respect to GIAB calls) as a function of coverage.** The full length reads were used for genotyping (blue), and additionally, reads were cut such as to cover at most two variants (red) and one variant (yellow) 38

2.8 **Confirming short-read variants.** We examine all distinct variants found by our method, GIAB high confidence, GATK/HC, and FreeBayes. Raw variant counts appear on top of each section, and the percentage of total variants is shown at the bottom. a All variants. b Variants in GIAB high-confidence regions. c Variants outside GIAB high-confidence regions 39

2.9 **Alignment matrix.** Here, the alphabet of possible alleles is the set of DNA nucleotides, i.e., $\Sigma = \{A, C, G, T\}$ 48

2.10 **Example graph.** Left: An alignment matrix. Right: The corresponding directed graph representing the bipartitions of active rows and active non-terminal rows, where the labels of the nodes indicate the partitions, e.g. '1,2 / .' is shorthand for $A = (\{1, 2\}, \{\})$.) 51

2.11 **Genotyping HMM.** Colored states correspond to bipartitions of reads and allele assignments at that position. States in C1 and C2 correspond to bipartitions of reads covering positions 1 and 2 or 2 and 3, respectively. In order to compute genotype likelihoods after running the forward-backward algorithm, states of the same color have to be summed up in each column 54

2.12 **The merger of two read partitioning HMMs with the same number of columns.** Top and middle: two HMMs to be merged; bottom: the merged HMM. Transition and emission probabilities not shown 60

3.1 **Assembly pipeline.** An overview of an example haploid consensus assembly pipeline which uses multiple sequencing technologies and pieces of software. This is non-exhaustive, as additional sequencing types and software could be included. 68

3.2 **Nanopore sequencing data.** a, Throughput in gigabases from each of three flow cells for 11 samples, with total throughput at top. Each point is a flow cell. b, Read N50 values for each flow cell. Each point is a flow cell. c, Alignment identities against GRCh38. Medians in a–c shown by dashed lines, dotted line in c is the mode. Each line is a single sample comprising three flow cells. d, Genome coverage as a function of read length. Dashed lines indicate coverage at 10 and 100 kb. HG00733 is accentuated in dark blue as an example. Each line is a single sample comprising three flow cells. e, Alignment identity for standard and RLE reads. Data for HG00733 chromosome 1 flow cell 1 are shown (4.6 Gb raw sequence). Dashed lines denote quartiles. 78

3.3 **Assembly metrics for Shasta, Wtgdb2, Flye and Canu before polishing.** a, NGx plot showing contig length distribution. The intersection of each line with the dashed line is the NG50 for that assembly. b, NGAx plot showing the distribution of aligned contig lengths. Each horizontal line represents an aligned segment of the assembly unbroken by a disagreement or unmappable sequence with respect to GRCh38. The intersection of each line with the dashed line is the aligned NGA50 for that assembly. c, Assembly disagreement counts for regions outside centromeres, segmental duplications and, for HG002, known SVs. d, Total generated sequence length versus total aligned sequence length (against GRCh38). e, Balanced base-level error rates for assembled sequences. f, Average runtime and cost for assemblers (Canu not shown). 81

3.4 **Shasta MHC assemblies compared with the reference human genome.** Unpolished Shasta assembly for CHM13 and HG00733, including HG00733 trio-binned maternal and paternal assemblies. Shaded gray areas are regions in which coverage (as aligned to GRCh38) drops below 20. Horizontal black lines indicate contig breaks. Blue and green describe unique alignments (aligning forward and reverse, respectively) and orange describes multiple alignments. 86

3.5 **Polishing assembled genomes.** a, Balanced error rates for the four methods on HG00733 and CHM13. b, Row-normalized heatmaps describing the predicted run lengths (x axis) given true run lengths (y axis) for four steps of the pipeline on HG00733. Guppy v.2.3.3 was generated from 3.7 Gb of RLE sequence. Shasta, MarginPolish and HELEN were generated from whole assemblies aligned to their respective truth sequences. c, Error rates for MarginPolish and HELEN on four assemblies. d, Average runtime and cost. 89

3.6 **HiRise scaffolding for 11 genomes.** a, NGx plots for each of the 11 genomes, before (dashed) and after (solid) scaffolding with HiC sequencing reads, GRCh38 minus alternate sequences is shown for comparison. b, Dot plot showing alignments between the scaffolded HG00733 Shasta assembly and GRCh38 chromosome scaffolds. Blue indicates forward aligning segments, green indicates reverse, with both indicating unique alignments. 94

4.1 **Pipeline for Comparative Annotation Toolkit.** 111

4.2 **Primate phylogenetic tree.** Taken from Rogers and Gibbs 2014 [1]. Evolutionary relationships among primates are shown. 117

4.3 **TransMap and Iso-Seq mappability in Mmul 10 compared to Mmul 8.0.1.** Comparative Annotation Toolkit (CAT) was used to project transcripts from GRCh38 to Mmul 10 and Mmul 8.0.1. Alignment coverage and identity were compared for orthologous transcripts found in each assembly pair. Additionally, Iso-Seq transcripts were mapped to both Mmul 10 and Mmul 8.0.1. (A) The box plots show the percentage change in identity and coverage of the TransMap alignments (left, middle), and coverage of Iso-Seq alignments (right) between Mmul 10 and Mmul 8.0.1. Transcripts with unchanged metrics were omitted from the plot. (B) Number of Iso-Seq transcripts that had changes in coverage between the assemblies. (C,D) Number of TransMap transcripts that had changes in coverage and identity between Mmul 10 and Mmul 8.0.1. 120

4.4 **Novel genes and gene models in the rhesus macaque.** (A) A novel gene model with homology to the cytochrome p450 protein family is predicted by the AugustusPB mode of the Comparative Annotation Toolkit (CAT). The gene structure and protein domain architecture of three isoforms are shown (top). The predictions arose from supporting Iso-Seq reads from five tissues (middle). Orthologous novel genes are also predicted in marmoset, orangutan, and gorilla assemblies; a protein alignment (bottom) of those genes along with a human CYP2C18 protein is shown. (B) Two macaque isoforms in ELN (tropoelastin) are predicted by the AugustusPB mode of CAT and are supported by macaque Iso-Seq data but differ significantly from human by two exons. The gene structure and functional domains for the last seven exons of this gene are shown (top), along with a comparison to a human transcript model. These two protein-encoding exons are also observed in marmoset, owl monkey, and mouse but not in apes, as a result of an ape-specific deletion (bottom) that changed the gene structure of tropoelastin. 122

4.5 **A bonobo isoform of ANAPC2 contains a novel exon predicted by the AugustusPB mode of CAT, which is supported by bonobo Iso-Seq data from iPSC tissue.** The exon is not seen in the human or chimpanzee annotations. a, The exon structure of this gene is shown for human, chimpanzee, and bonobo. The novel exon is alternatively spliced, seen in one isoform of ANAPC2 in bonobo and not the other. b, A sample of Iso-Seq reads from the bonobo iPSC tissue shows support for alternative splicing in ANAPC2. c, A protein alignment of the gene is shown for human, chimpanzee, and bonobo. 125

6.1 **Assembly continuity, phasing and base call accuracy metrics.** a, Contig NG50 values. b, Scaffold NG50 values. c, Haplotype phase block NG50 values. d, QV base call accuracy; as an example, QV60 is about one error per megabase. The dashed lines separate the assemblies into the four major categories as described in Table 1. The colours designate the type of haplotype phasing performed: Trio phasing using parental data, endogenous phasing using self-data, partial endogenous phasing, merging of haplotypes, and final references with various phasing approaches. The grey shaded regions in b are not applicable for scaffold metrics, as these are contig-only assemblies; however, the Flye assembler inserts gaps into contigs where there is uncertainty of a repeat sequence, and the $purge_dups$ function applied to the HiCanu contigs removes false duplications within contigs and creates a gap in the removed location. The grey shading in c indicates not applicable for phase blocks, because GRCh38 has many haplotypes and CHM13 is from a haploid (hap) cell line. The numbers in parentheses along the x axis are the assembly numbers. alt, alternate; mat, maternal; pat, paternal; phap, psuedo-haplotype; pri, primary; std., standard ONT read length; S-seq., Strand-Seq; UL., ultra-long ONT read length. 138

6.2 **Gene mapping in the pangenome graphs.** The first three show the percentage of protein-coding genes from GENCODE v38 able to be mapped in the gene annotation sets from Ensembl, CAT run on the MC graph based on GRCh38, and CAT run on the PGGB graph. The second three show the percentage of non-coding genes from GENCODE v38 able to be mapped on the same annotation sets. 143

6.3 **Transcript mapping in the pangenome graphs.** The first three show the percentage of protein-coding transcripts from GENCODE v38 able to be mapped in the gene annotation sets from Ensembl, CAT run on the MC graph based on GRCh38, and CAT run on the PGGB graph. The second three show the percentage of non-coding transcripts from GENCODE v38 able to be mapped on the same annotation sets. . 143

6.4 **Genes with frameshift mutations or nonsense mutations in the pangenome graphs.** A) Fraction of canonical transcripts with a frameshifting indel from GENCODE v38 in the Ensembl annotations and the CAT annotations of the GRCh38-based pangenome graph.B) Fraction of canonical transcripts with early in-frame stop codons in the Ensembl annotations and the CAT annotations of the GRCh38-based pangenome graph. 144

6.5 **Ensembl mapping pipeline results**. Percentages of protein-coding and non-coding genes and transcripts annotated from the reference set in each of the HPRC assemblies. Orange points represent T2T-CHM13 for comparison. 146

6.6 **The top 25 most commonly CNV genes or gene-families in the HPRC/HPRC+ assemblies, ordered by the number of samples with additional copies relative to GRCh38.** Grey bars represent the number of samples with additional copies. Blue circles represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. 151

6.7 **The top 30 most individually CNV genes or gene families in the HPRC/HPRC+ assemblies, ordered by total number of additional copies observed.** Blue circles again represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. The Grey bars represent the total number of additional copies summed over the samples. 153

6.8 **Transcriptome annotation of the assemblies.** A) Ensembl mapping pipeline results. Percentages of protein-coding and non-coding genes and transcripts annotated from the reference set in each of the HPRC assemblies. Orange points represent T2T-CHM13 for comparison. B) Assembled gene duplications per genome. The number of genomes containing a duplicated gene for 1529 protein-coding gene duplications indexed by increasing copy number, observed in the predicted reliable regions of the HPRC/HPRC+ genomes. C) Number of distinct duplicated genes or gene families per phased assembly relative to the number of duplicated genes annotated in GRCh38 (152). The GRCh38 gene duplications reflect families of duplicated genes, while the counts in other genomes reflect gene duplication polymorphisms. The assemblies are color coded according to their population of origin. D) The top 25 most commonly CNV genes or gene-families in the HPRC/HPRC+ assemblies, ordered by the number of samples with additional copies relative to GRCh38. Grey bars represent the number of samples with additional copies. Blue circles represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. E) The top 30 most individually CNV genes or gene families in the HPRC/HPRC+ assemblies, ordered by total number of additional copies observed. Blue circles again represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. The Grey bars represent the total number of additional copies summed over the samples. F) Dotplot illustrating haplotype-resolved GPRIN2 gains in the HG01361 assembly relative to GRCh38. G) Dotplot illustrating SPDYE2/SPDYE2B haplotype resolved gains within a tandem duplication cluster of the HG00621 assembly relative to GRCh38. 154

B.1 **Precision and Recall** (Top) of MarginPhase on PacBio and WhatsHap on Nanopore data sets in GIAB high confidence regions. **Genotype Concordance (Bottom)** (wrt. GIAB high confidence calls) of MarginPhase (mp, top) on PacBio and WhatsHap (wh, bottom) on Nanopore. 181

B.2 **Genotyping Errors** (wrt. to GIAB calls) as a function of coverage. The full length reads were used for genotyping (blue) and additionally, reads were cut such as to cover at most two variants (red) and one variant (yellow). 182

B.3 **Read Depth: PacBio** Precision, Recall, and F-Measure as a function of depth. . . 186

B.4 **Read Depth: Nanopore** Precision, Recall, and F-Measure as a function of depth. 187

B.5 Genotyping concordance as a function of genotyped variants for different thresholds (0, 20, 50, 80, 100, 150, 200, 300, 400, 500) on the genotype quality. Since the maximum quality value output by MarginPhase is limited to 100, thresholds larger than 100 are not considered in the plots. 192

B.6 Observed genotype concordance as a function of the expected genotype concordance of the variant calls. Dot sizes correspond to the number of calls which were reported with the same quality score. 193

C.1 Markers aligned to a run length encoded read. 213
C.2 A marker alignment represented as a dot-plot. Elements that are identical between the two sequences are displayed in green or red - the ones in green are the ones that are part of the optimal alignment computed by the Shasta assembler. Because of the much larger alphabet, matrix elements that are identical between the sequences but are not part of the optimal alignment are infrequent. Each alignment matrix element here corresponds on average to a 13×13 block in the alignment matrix in raw base sequence. 214
C.3 An example of a portion of the read graph, as displayed by the Shasta http server. . 222
C.4 An example of a portion of the read graph showing obviously incorrect connections . 223
C.5 A marker graph representing a single read. 224
C.6 An illustration of marker graph construction for two sequences. 225
C.7 (A) A marker graph with linear sequence of edges colored. (B) The corresponding assembly graph. Colors were chosen to indicate the correspondence to marker graph edges. 227
C.8 (A) A simple bubble. (B) A superbubble. 234
C.9 A) An example POA, assuming approximately 30x read coverage. The backbone is shown in red. Each non-source/sink node has a vector of weights, one for each possible base. Deletion edges are shown in teal, they also each have a weight. Finally insertion nodes are shown in brown, each also has a weight. (B) A pruned POA, removing deletions and insertions that have less than a threshold weight and highlighting plausible bases in bold. There are six plausible nucleotide sequences represented by paths through the POA and selections of plausible base labels: G;AT;A;T;A;C:A, G;AT;A;T;A;C:G, G;A;T;A;C:A, G;A;T;A;C:G, G;A;C:A, G;A;C:G. To avoid the combinatorial explosion of such enumeration we identify subgraphs (C) and locally enumerate the possible subsequences in these regions independently (dotted rectangles identify subgraphs selected). In each subgraph there is a source and sink node that does not overlap any proposed edit. 248
C.10 **Visual representation of run length inference.** This diagram shows how a consensus run length is inferred for a set of aligned lengths (X) that pertain to a single position. The lengths are factored and then iterated over, and log likelihood is calculated for every possible true length up to a predefined limit. Note that in this example, the most frequent observation (4bp) is not the most likely true length (5bp) given the model. 250
C.11 **MarginPolish Images** A graphical representation of images from two labeled regions selected to demonstrate: the encoding of a single POA node into two run-length blocks (i), a true deletion (i), and a true insert (ii). The y-axis shows truth labels for nucleotides and run-lengths, the x-axis describes features in the images, and colors show associated weights. 255

C.12 The sequence-to-sequence model implemented in Helen. 259
C.13 Run-length confusion in different versions of Guppy base caller 265

D.1 Size distribution of structural variants (¿50 bp) extracted from the Shasta assembly graph for HG002 and the structural variants in the Genome In A Bottle (GIAB) catalog for the same sample. a) Full size distribution for deletions (top) and insertion (bottom), in log-scale. b) and c) zoom in the two peaks caused by Alu (˜300 bp) and L1 (˜6 Kbp) insertion polymorphisms. 280
D.2 Log frequency of each run length as found in the GRCh38 reference for all bases A,C,G,T up to 100bp. Run lengths greater than 15 account for approximately 0.012% of all homopolymer runs in GRCh38. 289
D.3 Contig NGx for CHM13 Shasta-HELEN nanopore assembly vs Canu CCS (HiFi) assembly 299
D.4 Contig NGAx for CHM13 Shasta-HELEN nanopore assembly vs Canu CCS (HiFi) assembly 300
D.5 Dotplot for the scaffolded HG002 assembly, aligned with GRCh38. Blue dots represent unique alignments and orange dots represent repetitive alignments. 301

List of Tables

2.1 Distribution of candidate novel variants across different regions of interest. All variants refers to the variants in the Long Read Variants set, and Novel Variant Candidates are those described in Section 3.6. 42

3.1 CAT transcriptome analysis of human protein coding genes for HG00733 and CHM13. 91

5.1 Comparison of GRCh38 and T2T-CHM13v1.1 human genome assemblies. GRCh38 summary statistics exclude "alts" (110 Mbp), patches (63 Mbp), and chromosome Y (58 Mbp). The number of exclusive genes or transcripts is as follows: for GRCh38, GENCODE genes and transcripts not found in CHM13; and for CHM13, extra putative paralogs that are not in GENCODE. 130

5.2 Table: GRCh38 Comparison. Excl = Exclusively found in each assembly. 132

B.1 Switch error rates of MarginPhase and WhatsHap for each chromosome inside of the GIAB high confidence regions. 183

B.2 Results from re-typing GIAB truth set indels using WhatsHap on the PacBio and Nanopore reads. 184

B.3 Summary of genotyping results on homozygous variants. 188

B.4 Summary of genotyping results on heterozygous variants. 189

C.1 Description of trained models for HELEN. 264

D.1 QUAST assembly metrics of three samples on four assemblers before polishing, compared against GRCh38 with no alternate contigs. 279

D.2 QUAST disagreement count for four assemblers on different regions of the genome for four samples. We report disagreements that happen in all chromosomes of GRCh38, then incrementally exclude centromeric regions, segmental duplication regions (Seg Dups), and all other regions enriched for SVs (chrY, acrocentric chromosome arms, and QH-regions) 281

D.3 Disagreement count in the intersection of the assemblies for each sample (see Online Methods). Total Disagreements describes all disagreements found in 100bp windows before taking the intersection; note that these counts are very close to those reported by QUAST. Consensus Disagreements describes disagreements in the intersection of the four assemblies. Genome fraction describes total coverage over GRCh38 for the consensus sequence. 281
D.4 Disagreement count and fraction of genome covered on chromosome X for four assemblers on CHM13 assemblies with no polishing, compared to the chromosome X assembly from the Telomere-to-Telomere Consortium. These numbers were obtained via running QUAST. 282
D.5 BAC analysis on selected dataset. BACs were selected (31 of CHM13 and 16 of HG00733) for falling within unique regions of the genome, specifically >10 Kb away from the closest segmental duplication. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus in the assembly. *Attempted* refers to the number of BACs which have an alignment for >5 Kb of sequence with >90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs. 282
D.6 BAC analysis on full dataset, 341 on CHM13 and 179 on HG00733. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus. *Attempted* refers to the number of BACs which have an alignment for ¿5Kb of sequence with ¿90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs. 282
D.7 BAC analysis intersection of attempted BACs by all four assemblers, 65 on CHM13 and 27 on HG00733. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus. *Attempted* refers to the number of BACs which have an alignment for > 55Kb of sequence with > 90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs. 283
D.8 Base-level accuracies on four different assemblers for three samples. Analysis is performed with whole-genome truth sequences. 283
D.9 Base-level accuracies on four different assemblers for three samples in the regions of intersection of the assemblies. Analysis is performed only on regions where all assemblers have an assembled sequence. 284
D.10 Runtime and cost of three assembly workflows on Amazon Web Services (AWS) platform. 284
D.11 Runtime breakdown for each step of the Shasta assembler. 285
D.12 Structural variants extracted from HG002 assembly graph compared to GIAB SV set in high-confidence regions. 285
D.13 CHM13 MHC unpolished Shasta assembly as compared to the nearest matching haplotype in hg38 (GL000251.2) 286
D.14 QUAST results for the HG00733 trio-binned maternal reads, using all four assemblers. 286
D.15 HG00733 Maternal trio binned MHC unpolished Shasta assembly as compared to the nearest matching haplotype in hg38 (GL000255.1) 287

D.16 Base-level accuracies comparing Racon & Medaka and MarginPolish & HELEN pipelines on Shasta assemblies for three samples. Analysis is performed with whole-genome truth sequences. 288
D.17 QUAST results for the Shasta assemblies for all samples, post polishing with MarginPolish-HELEN. 289
D.18 Base-level accuracies comparing Racon & Medaka and MarginPolish & HELEN pipelines against CHM13 Chromosome-X. The truth Chromosome-X sequence used reflects the most accurate haploid truth sequence available. 289
D.19 Base-level accuracies improvements with MarginPolish and HELEN pipeline on four different assemblers for two samples. Analysis is performed with whole-genome truth sequences. 290
D.20 Single-chromosome error rates after polishing with short reads. 10X Chromium reads for sample CHM13 were used to polish via Pilon polishing software. The top half of the table shows the results of three rounds of Pilon, starting from the CHM13 Shasta chrX assembly that had been polished with MarginPolish and HELEN. The bottom half shows the results of three rounds of Pilon, starting from the raw Shasta assembly. 290
D.21 Runtime and cost of two polishing workflows on Amazon Web Services (AWS) platform. 291
D.22 Runtime and cost of two polishing workflows run on a 29 Mb contig from the HG00733 Shasta assembly. MarginPolish uses an improved stitch method not used in original runs and Racon was run once instead of four times as was done in the full runs. All runs were configured to use 32 CPUs, except for the GPU runs which were performed with 16 CPUs and 1 GPU (Tesla P100). 292
D.23 Transcript-level analysis with Comparative Annotation Toolkit (CAT) of MarginPolish & HELEN and Racon & Medaka on three samples from Shasta assemblies. 293
D.24 Gene-level analysis with Comparative Annotation Toolkit (CAT) of MarginPolish & HELEN and Racon & Medaka on three samples from Shasta assemblies. 294
D.25 Transcript-level analysis with Comparative Annotation Toolkit (CAT) of four HG00733 assemblies polished with MarginPolish and HELEN. 295
D.26 Gene-level analysis with Comparative Annotation Toolkit (CAT) of four HG00733 assemblies polished with MarginPolish and HELEN 296
D.27 BUSCO results of three samples using two polishing workflows on Shasta assemblies. 296
D.28 BUSCO results for four assemblers on HG00733, post polishing with MarginPolish and HELEN. 297
D.29 CHM13 QUAST results for Shasta, MarginPolish, HELEN and PacBio HiFi assembly. Stratified disagreement counts were added after manual determination. 298
D.30 Disagreement count in the intersection of the assemblies between the PacBio-HiFi and the Shasta assembly of CHM13. Total Disagreements is all disagreements found in 100bp before windows before taking the intersection, note it is very close to that reported by QUAST. Consensus disagreements: Disagreements in the intersection of the four assemblies. 299
D.31 CHM13 Chromosome-X error rate analysis with Pomoxis for Shasta, MarginPolish, HELEN, and PacBio HiFi assembly. 299

D.32 QUAST results for all 11 Shasta assemblies scaffolded with HiRise, post polishing with MarginPolish-HELEN 301

Abstract

Using long reads to improve haplotype phasing, genome assembly, and gene annotation

by

Marina Haukness

Despite their accuracy, next-generation DNA sequencing technologies have limited utility in analyzing ambiguous and repetitive parts of the genome due to the short length of reads. Third-generation long read DNA sequencing technologies, such as those from Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PacBio), allow us to explore much more of the genome and perform more comprehensive genomic analyses. However, new software must be developed for these analyses in order to take advantage of the increased read lengths, while mitigating errors from base-level inaccuracies. In this thesis, I explore the advantages of long reads for haplotype phasing and genome assembly. I then use genome assemblies created from long reads to perform comparative genomics analyses, focusing on gene annotation of new, high-quality assemblies of primates and humans, including annotating the first fully complete human genome and a human pangenome containing over 90 distinct haplotypes.

To all those who I consider my family, both biological and chosen.

Acknowledgments

When I started graduate school at UC Santa Cruz as a M.S. student, many of my friends and family were convinced that I would end up getting a Ph.D. Turns out, they were right. However, it was a bumpy road, and I wouldn't have made it to this point without the support of many people.

I came to graduate school after becoming disillusioned by software engineering while majoring in computer science and math as an undergrad, when I decided I wanted to use those skills to work on something more meaningful and address my interest in biology that I had never fully explored. The first person I have to thank is my advisor, Benedict Paten, who made it possible to do this. The work the lab was doing when I first joined was so exciting, and it has only gotten more exciting along the way. Benedict was enthusiastic about my background and has helped me get involved with so many major, meaningful collaborations, including many large and impressive research consortia like the T2T and HPRC. His guidance was critical throughout my time in the program.

I also have to thank my other committee members, Richard "Ed" Green and David Haussler. In particular, it was great to work with Ed and the rest of the Paleogenomics lab to get exposed to research on non-human animals to start off the portions of my thesis involving comparative genomics. I am also grateful to the broader BME program at UCSC. The faculty are wonderful and are all pioneers in their respective fields, and the other BME students are so inspiring, both in their research and outside interests. I'd also like to specifically acknowledge everyone involved in the unions, par-

ticularly those who have gone on strike multiple times now, who helped fight for a living wage so us graduate students can survive in an area with an increasingly high cost of living.

My past and present lab mates in the Computational Genomics Lab are brilliant and have become some of my closest friends. Thank you to Trevor Pesout, Kishwar Shafin, Ryan Lorig-Roach, Xian Chang, Jean Monlong, Jordan Eizenga, Mobin Asri, Melissa Meredith, Mira Mastoras, James Casaletto, Robin Rounthwaite, Cecilia Cisar, Nick Keener, Stephen Hwang, Jimin Park, Charlie Markello, Adam Novak, Jouni Siren, Andrew Bailey, Colleen Bosworth, Ian Fiddes, Joel Armstrong, John Vivian, and Jonas Sibbesen. Working on research with all of you was as pleasant of an experience as it could possibly be. Of course, we had fun outside of work too, and I have many fond memories of drinking beer, having picnics in the park and on the beach, eating baked goods, playing Pokemon go (on two phones at the same time!), buying inflatable couches for windowless grad caves, going on runs, hiking, training for a Tough Mudder, among many other things. Also, thanks to others associated with the lab: Karen Miga, Julian Lucas, Mark Diekhans, and Miten Jain. Each of you has been tremendously influential during my Ph.D.

Additionally, I am grateful for the opportunity to work with so many other amazing, talented people outside of the Computational Genomics Lab. This field is incredibly collaborative and I had the pleasure of being introduced to so many people outside of my own institution, particularly through consortia such as the Telomere-to-Telomere (T2T) Consortium, and the Human Pangenome Reference Consortium

(HPRC). Some people in particular to thank are Jana Ebler and Tobias Marschall, Paolo Carnevali, Arang Rhie, Evan Eichler, Wes Warren, and Mark Chaisson, among too many others to count.

I dedicated this dissertation to everyone I consider to be a part of my family. First and foremost, this includes my parents, who from a young age emphasized the importance of learning and the joy that can be found pursuing it, and have been so supportive throughout my entire life and education journey. This also includes my brothers, Andy and Alex, who I have grown to actually like instead of tolerate over the years, particularly after we all independently discovered that we like running. This is not the end of who I consider my family, though. I was fortunate to find multiple other families who viewed me as another member of theirs. This includes the Burke family: Kristine, Jeff, and Lauren (and Rosie!), who still make sure I am fed on a weekly basis. They have opened their home to me so many times over the years, perhaps most notably in August 2020 when I was forced to evacuate from my residence for over a week due to the forest fires in the Santa Cruz mountains. Also, thank you to the Hembrough family: Judi, Allie, and the late Roger (and Bayley!), who have also been there for me since high school, welcoming me into their home whenever needed, and sharing a glass of chardonnay whenever I wanted.

I am fortunate to consider a large number of people to be my friends, many named at various points above. I also would like to thank Anh, Jazmin, Laura, Nava, Sophie, Ashka, Lisa, Akshar, Sam, Dave, Brandon, and Alex for being there for me over the years. On the other hand, there are unfortunately far too many people who were

here to see me start this program, but are no longer here to witness me finish it. These include my friend Willie Zuniga, my great-aunt Sharon Faleide, my grandma Edith "Grand-E" Szatai, my grandpa Bill Roeszler, and Roger "Rogie Rog" Hembrough. I hope I am making you proud.

Last but not least, anyone who has ever talked to me for any time at all probably knows how obsessed I am with my pets, and I of course need to thank them all: Charmy, my bearded dragon, started grad school with me, but unfortunately passed away a couple years in. Gremlin, or Grem for short, my crested gecko, who has the cutest eyelashes a gecko could possibly have. And my two cats, Taco and a cat-who-is-still-somehow-not-officially-named (but goes by Baby Girl, Girl Cat, Bagel, Tuesday, Noodle, among many other things). Without them I don't know how I would have survived the pandemic and the whole end of grad school.

I'm not always the best at vocalizing the appreciation I have for all of these important people in my life, but I hope you know that I am grateful for you all.

Chapter 1

Introduction

At their core, many genomic analyses aim to determine an individual's underlying genomic sequence, which requires the use of accurate DNA sequencing methodologies. Over the past two decades, there have been frequent and substantial innovations in DNA sequencing technologies that have transformed the field of genomics, increasing the quality of analyses and types of possible analyses that can be done at lower costs and higher speeds. The cost of DNA sequencing and subsequent genome reconstruction has decreased dramatically; for example, the cost of sequencing the human genome dropped from $300 million in 2001 for the first human genome in the Human Genome Project to around $1000 in 2020, with the biggest drop in cost in 2008 resulting from the introduction of next-generation sequencing, which produces high-quality, short reads. This has made genome sequencing for personal health diagnoses, such as cancers and other genetic diseases, increasingly feasible. It has also made it easier to study more nonhuman species, giving us a more complete picture of the genomic diversity of organisms

living on this planet, and more insight into processes like evolution.

After the introduction of next-generation sequencing, many additional types of DNA sequencing have emerged, each with characteristics that are suited for different kinds of genomic analyses. There are now multiple technologies available for sequencing longer sections of DNA at once, which result in what are referred to as long reads. These reads, though less accurate than their predecessors, allow for analyses that were previously impossible due to the short read lengths of next-generation reads. Much of this dissertation will focus on these longer sequencing reads and how they have transformed the field.

Short next-generation reads, such as those from Illumina, are highly accurate at a per-base level, cost-effective, and allow for a wide range of genomics analyses to be performed. Their sequencing errors are well-characterized and easy to model statistically. These properties make next-generation reads ideal for genotyping small variants in well-characterized regions of the genome. Despite their accuracy, short reads suffer from being more difficult to map to the genome, especially in repetitive regions such as LINE and SINE elements, segmental duplications, centromeres, and telomeres. This is because sequencing reads that arise from repetitive genomic sequences that have a length greater than the length of the read often cannot be unambiguously mapped to the read's precise origin. This results in short reads being able to map to only about 90% of the current reference genome assembly [2], and even less (80%) at high confidence [3]. Any genomics analyses that rely on a read mapping step, such as variant calling, cannot confidently be completed in these difficult-to-map regions. Short reads also have limited

utility when assembling a genome *de novo* (from scratch), as it can be hard to find long enough overlaps between reads to unambiguously extend assembled segments, resulting in assemblies that are very discontinuous, with many short assembled fragments and few long ones. Additional problems with short reads include uneven read coverage and sequencing bias [4].

Long read sequencing technologies, such as those from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), have proliferated in recent years and offer both new solutions and new challenges to genomics analyses. Long read sequencing helps address some of the problems that arise from traditional short read sequencing methods, such as improving ambiguous read mapping in repetitive regions, because their length can span all or most of many repeats, and can include more of the unique flanking sequence on either end. Unfortunately, these reads have some downsides. The sequencing error rate can be close to 10%, and errors are not distributed randomly, and instead are concentrated in homopolymers, where a single base of DNA is repeated any number of times. Long read sequencers also tend to have lower throughput and cost more than short reads [5]. Despite these downsides, long reads have already transformed the field of bioinformatics, and their accuracy and throughput are quickly increasing. Some candidate problems that long reads are well-suited for include haplotype phasing and genome assembly, which are two areas I have focused on in this dissertation. Long reads can also be used in RNA-sequencing, and help improve isoform resolution among transcripts of genes, and paralog resolution within homologous gene families, which I took advantage when performing gene annotation on various assemblies.

Once higher quality genome assemblies are made, they will allow for studying regions of the genome previously almost entirely ignored, such as the centromeres, which are notoriously difficult to assemble due to their highly repetitive structure, as well as any of the other repetitive regions of the genome. This will also allow for higher quality annotations made on top of the assemblies, such as repeat annotations and gene annotations. Having higher quality assemblies and annotations for more species, and for more individuals within a species, will then pave the way for more interesting and complex comparative genomics analyses. Then we can gain more insight into the evolutionary histories of these species, particularly within such challenging regions, as well as more knowledge about the variation present in these regions among individuals within a species.

The past six years of graduate school have been an exciting time to be involved in the field, where the state-of-the-art changes on a yearly basis. This thesis describes many projects I have been involved in, which may seem fairly unrelated on their surface, but one unifying theme amongst them all is that long reads can be used to improve many different genomic analyses. The first part of my dissertation focused on the problem of haplotype phasing. I worked on developing software called MarginPhase, which uses a Hidden Markov Model (HMM) to separate an input set of long reads into two haplotypes, and then identify variants within each haplotype to produce a set of phased variants. I then switched focus to genome assembly. I ran many different assembly pipelines using different types of long reads, and also gained expertise at the use of long range contact information within proximity ligation reads to string smaller assembled sequences of

DNA (contigs) into bigger scaffolds. I analyzed the quality of these assemblies along many different metrics. This culminated in the publication of Shasta, an assembler that uses long nanopore reads from ONT. I then switched focus to gene annotation and comparative gene analysis on these assemblies made from long reads. I helped annotate and analyze two new primate genome assemblies, the rhesus macaque and the bonobo, using long RNA sequencing reads to aid in the identification of novel genes and novel transcript isoforms. I joined the Telomere-to-Telomere Consortium to help lead the gene annotation on the first fully complete human genome assembly, identifying genes in regions of the genome that had never been assembled before, such as regions in and near the centromeres. I joined the Human Pangenome Reference Consortium and annotated the genes on a pangenome made of 47 new diploid assemblies for a diverse panel of humans, and focused on gene families experiencing copy number variation within this panel.

The rest of my thesis will focus on my contributions of each of these projects. Because each of these sections are quite distinct from the others, they require a lot of background information, and rather than include that all up front, relevant background can instead be found at the beginning of the corresponding chapter. However, a small amount of background relevant to all chapters, mainly regarding sequencing technologies, can be found after this introduction. The rest of the document is organized as follows: Chapter 2 is about the development and use of MarginPhase, a diplotyper that uses an HMM to simultaneously genotype and phase haplotypes using long DNA sequencing reads. Chapter 3 is about using long nanopore reads for genome assem-

bly, and development of a pipeline to fully sequence and assemble 11 genomes in 11 days. Chapter 4 is about comparative genomics, specifically gene annotation on primates, focusing on two new assemblies of the rhesus macaque and bonobo and detailing some software contributions made to the Comparative Annotation Toolkit. Chapter 5 is about gene annotation on the first fully complete, telomere-to-telomere genome assembly as a part of the Telomere-to-Telomere (T2T) consortium. Chapter 6 is about gene annotation on a human pangenome as a part of the Human Pangenome Reference Consortium (HPRC). The appendices include mainly supplemental information from the corresponding papers.

Background

The following section details some additional background that is largely relevant to each of the remaining chapters within my dissertation.

DNA sequencing techniques

This dissertation focuses on analyses that use whole-genome sequencing, which aims to sequence the entire genome for a given sample. This is in opposition to targeted sequencing, which aims to sequence only selected portions of the genome, as well as microarray-based methods, which serve to identify the bases at predetermined loci within the genome. Whole-genome sequencing is immensely powerful, as it does not require predetermining loci of interest within which to restrict sequencing or genotyping.

Sanger sequencing

Sanger sequencing was one of the first methods for DNA sequencing, developed in 1977 by Frederick Sanger [6], and remained one of the most widely used sequencing technologies for many years. The current iteration of Sanger sequencing involves the use of modified di-deoxyucleotide triphosphates (ddNTPs) which will terminate the construction of a DNA chain after they are incorporated. For a given DNA sample, four separate reactions occur that involve a mix of regular deoxynucleotide phosphates (dNTPs) and one of the ddNTPs. Gel electrophoresis is used to read the results of the chain reactions. The reads produced by this process can be up to 1000 base pairs long and are extremely accurate, with accuracies of approximately 99.999%. Unfortunately,

the process does not scale well to generate the large amounts of data needed for genomics projects today, but it still has its uses for smaller-scale needs.

Next-generation sequencing

The next type of sequencing technologies developed after Sanger sequencing are referred to as "next-generation sequencing" (or "second-generation sequencing", "shotgun sequencing", or "short read sequencing"). Today, this term is mostly used to refer to Illumina sequencing [7], which is how the rest of this dissertation will use the term as well. These techniques follow the general workflow of fragmenting the DNA within a sample, ligating adapters, amplifying the DNA with a polymerase chain reaction (PCR), and sequencing the resulting DNA. The sequencing here is "sequencing by synthesis", where in each round, each DNA strand gets extended by a fluorescently labeled dNTP, and the base that was used is recorded.

There are additional types of Illumina sequencing such as paired end sequencing, where pairs of reads are produced that originated from the same DNA, and the linkage information between the pair can improve read mapping to a genome. This type of sequencing technique is highly parallelizable, and results in reads that are still highly accurate (approximately 99.9%, depending on machine), with read lengths between 50 to 400 base pairs, commonly 150 or 300bp. Though less accurate than Sanger sequencing, the throughput allows for quick production of millions of reads from a sample.

Proximity ligation sequencing

Chromosome conformation-capture (3C) was the first technique used to detect the frequencies of interactions between genomic loci [8]. 4C combines 3C techniques with high-throughput sequencing [9]. HiC can be considered a type of either of these methods, and it is the only type of chromosome capture technique that I used within the work for my dissertation. Each of these techniques follows the same general set of steps: cross-linking DNA that is in close physical proximity (the actual proximity ligation), then prepping that DNA for sequencing, and performing the sequencing. HiC uses Illumina next-generation sequencing for the final sequencing step. It results in sets of pairs of reads, where the pairs of reads are from portions of DNA that were in close physical proximity to each other during the initial cross-linking step. Each of these interactions can be plotted to create a contact map, which allows for the visualization of the density of these contacts across the genome. HiC data is particularly useful for scaffolding *de novo* genome assemblies. Contigs that have many HiC interactions between them are likely near each other along the chromosome, so they can be scaffolded together.

Linked reads

10X Genomics commercialized the production of linked sequencing reads. Unfortunately, this technique was discontinued in 2020, although these reads are mentioned in a couple sections within my dissertation, so they still deserve some background. A linked read is made up of multiple short reads that originate from a single, long piece of

DNA. Each short read is barcoded with a unique molecular barcode to designate they arose from the same original molecule, and then they are sequenced on a next-generation sequencer. The long range information allows these reads to be used for some of the analyses that long reads are good for, like structural variant calling and *de novo* genome assembly [10], though they are not quite as good as the long reads discussed next.

Long read sequencing

The next major innovation for DNA sequencing was deemed "third-generation sequencing", though it is also often referred to as "long read sequencing", as the reads produced can be much longer than any of the previously mentioned technologies (tens to hundreds of thousands of base pairs per read, even occasionally more than a million). However, the long read length comes with a cost of being less accurate. This space is dominated by two companies, Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT).

PacBio reads are generated from the SMRT (Single Molecule Real-Time) sequencing platform. For a piece of DNA, which can range in length up to tens of thousands of base pairs, PacBio makes a circular piece of DNA with the DNA insert and adapters on either end. The DNA is sequenced using fluorophores. The first type of reads from PacBio were Continuous Long Reads (CLR), and had insert sizes from 25kb to 100kb, but low accuracies (88-90%). Later, PacBio introduced their High Fidelity (HiFi) reads, that used Circular Consensus Sequencing (CCS) mode rather than CLR mode. These reads are much smaller in size (10-25 kb), which allows the polymerase

to read around the piece of circular DNA many times to generate multiple subreads for the DNA insert. A consensus sequence is called, and this leads to accuracies greater than 99%. As one may expect, these techniques lead HiFi reads to cost significantly more than other sequencing types.

ONT is responsible for commercialization of nanopore sequencing, which was initially conceived at UC Santa Cruz [11]. Nanopore sequencing involves sequencing DNA as it passes through a pore embedded within a membrane. To do so, DNA strands get adapters added, which include a motor protein to help pull them through the pore. An electric field is applied to the membrane, setting up a difference in charge on either end. The negatively charged DNA is pulled through to the positively charged side, passing through a pore on its way. The current is measured as the DNA passes through the pore, and the resulting signal is used to infer which nucleotides were present in the pore. Nanopore sequencing started off being wildly inaccurate, but accuracies have improved to be around 90%-94% these days. The majority of errors in nanopore reads are concentrated within homopolymers, which are stretches where a single nucleotide base repeats, as it is difficult to determine the number of bases involved from the way they pass through the pore.

RNA sequencing

All of the sequencing techniques mentioned up until this point have been for use on DNA. However, DNA gives only a static view about the genome of a cell– RNA is much more dynamic, and sequencing RNA can give more information about

things like an organism's gene expression across multiple tissue types. A portion of my thesis focuses on gene annotation, which directly benefits from the inclusion of RNA sequencing information, instead of having to rely completely on the genomic sequence to predict the existence of transcripts and genes.

RNA sequencing generally requires converting a single-stranded RNA molecule back into a complementary strand of DNA (cDNA), and then sequencing the cDNA with existing techniques. Next-generation sequencing is commonly used, so it has all the same advantages and disadvantages as discussed earlier. Another downside of these techniques relevant to transcriptome analysis is that short reads may be unable to distinguish between different isoforms of a gene, as they may not be long enough to cover all of the exons involved.

As one may hope, long reads can additionally be used to sequence RNA. Iso-Seq use PacBio reads to sequence full-length cDNA transcripts. Nanopore sequencing can also be used to sequence full-length cDNA transcripts, but it has the additional advantage of being able to sequence native RNA strands themselves. This adds the ability to identify any modifications to the RNA bases, whereas any method involving a conversion to cDNA loses these base modifications.

Chapter 2

Haplotype Phasing

Background

Variant identification

Identifying genetic variants within a genome can reveal valuable information about an individual or population, such as the genetic causes of clinically significant phenotypes or the ancestry of DNA [12]. There are several different types of genetic variation: single nucleotide variations (SNVs), where a single nucleotide differs at a locus, insertions and deletions (indels), where there is an insertion or deletion of a base (or more), and structural variations (SVs), where there are larger differences greater than 50 base pairs in length. In a diploid individual with two copies of each chromosome, variations can be homozygous, meaning the same on both copies of the chromosome, or heterozygous, meaning different on each copy. Variant identification comprises two related processes: genotyping and phasing. Genotyping refers to determining the indi-

vidual's genotype at each locus in the genome, and phasing refers to determining which variants arise from which haplotype and are therefore likely to inherited together (barring recombination). Figure 2.1 shows an example of the difference between genotypes and haplotypes in a diploid organism. To get a complete understanding of the genetic variation in an individual, both genotyping and phasing, which can together be called *diplotyping*, should be completed.

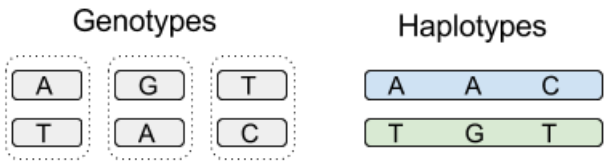

Figure 2.1: **An example of genotypes vs. haplotypes.** Genotypes describe which base are at a location in the genome. The genotypes at each of the three loci here are A/T, G/A, and T/C. Haplotypes designate which are inherited together on the same chromosome. In this example, the A/A/C variants are on the same chromosome and therefore the same haplotype, and the T/G/T variants are on the other.

Genotyping is often the only one of these processes undertaken after sequencing an individual, as it allows us to identify the existence of variations in someone's DNA. There are many existing variant callers that can determine genotypes based on genome sequencing information, commonly from accurate but short sequencing reads. Technology such as SNP chips make this simple genotyping even more straightforward and economical. Less attention is paid to the organization of the variants into haplotypes; however, this organization can lead to phenotypically distinct traits and therefore may be necessary to ascertain. An example of when haplotype phasing might be useful in addition to genotyping is a case where an individual has two point mutations that affect a gene, one in the promoter which causes a decrease in abundance of the protein,

and another in the protein coding region of the gene which results in a nonfunctional protein, such as a nonsense mutation leading to early truncation during translation. This example situation is pictured in Figure 2.2. If the individual is heterozygous for these mutations, meaning they have one functional wild type copy of both the promoter and the protein-coding gene, and one mutant copy of each, then there are two possible ways these mutations can be arranged across the two copies of the chromosome they are on. In the first arrangement, both wild type copies of the promoter and protein are on one chromosome, and both mutant copies are on the other chromosome. The individual would be able to produce a normal amount of functional protein in this case despite the two mutations. However, in the second possible arrangement, the mutant promoter and wild type protein are on one chromosome, and the wild type promoter and mutant protein are on the other. Here, the patient would not be able to produce enough of the functional protein, which could be clinically significant.

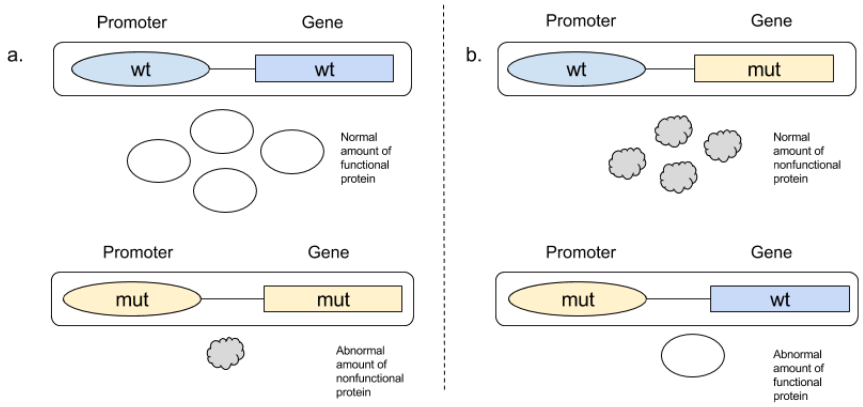

Figure 2.2: **An example of clinically significant differences in haplotypes.** Two different possibilities for organization of the same genotype. In (a), the wild type promoter and gene are on the same chromosome and are therefore able to produce a normal amount of functional protein. In (b), only a reduced amount of functional protein is produced.

An example of the challenges of diplotyping adjacent sites in practice is demonstrated in Figure 2.3. Three SNV positions are shown, with the true haplotypes in grey, and reads in red and blue coming from each haplotype. Without knowledge of the haplotypes, we could reasonably predict the genotypes for these sites as A/C, G/T, and G/C. However, due to the potential for sequencing errors, this may lead to wrong predictions, as the middle site actually had two sequencing errors in the reads covering it leading to an incorrect prediction instead of T/T. Errors like this can be avoided if we have knowledge of which haplotypes each read originated from, so we could then determine that there must have been sequencing errors on the second site.

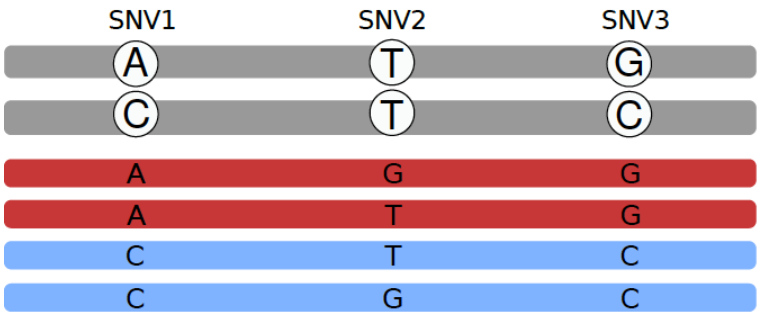

Figure 2.3: **Motivation and overview of diplotyping**. Gray sequences illustrate the haplotypes; the reads are shown in red and blue. The red reads originate from the upper haplotype, the blue ones from the lower. Genotyping each SNV individually would lead to the conclusion that all of them are heterozygous. Using the haplotype context reveals uncertainty about the genotype of the second SNV.

Existing variant calling and phasing tools and their limitations

Many existing genotyping pipelines are designed for short reads of DNA, where they excel at calling variants in regions where the reads can map with high confidence. At the time when this work was being completed in 2017-2018, some of the main vari-

ant callers that were used are GATK HaplotypeCaller, Freebayes, SAMtools mpileup, DeepVariant, and Clairvoyante [13, 14, 15, 16, 17]. Some of these variant callers, like GATK HaplotypeCaller, Freebayes, and SAMtools mpileup rely on statistical methods that model the error distributions found in short, Illumina reads, whereas DeepVariant and Clairvoyant use deep neural networks. These short read variant callers have some limitations, which is that they require the reads to be aligned to the reference genome of interest and short reads cannot be unambiguously mapped to all areas of the genome. Therefore, there are many regions within the genome upon which variants cannot confidently be called. Some of these variant calling tools claim to be able to be used with long reads as well, but the performance is not nearly as good when using error-prone longer reads. There are also some specialized variant callers, such as PoreSeq [18] and marginAlign [19] for Oxford Nanopore, and a method proposed by Guo et. al which works only on PacBio reads [20]. However, none of these programs leverage any linkage information of the long reads to attempt phasing of the variants.

Haplotype phasing is challenging with standard short-read sequencing technologies, because reads are not long enough to span multiple variants from which phasing constraints can be implied. For example, the human heterozygosity rate is approximately 0.1%, or one base out of every thousand differing, which means that reads that are tens of thousands of bases long are likely to contain multiple variants. This means that short reads like 150bp Illumina reads are not likely to span multiple variant sites, and are therefore not informative for phasing. Many existing approaches for phasing are statistical methods, and some make use of information in related individuals to better

impute possible haplotypes [21]. Long reads are particularly well-suited for haplotype phasing, because longer reads more frequently contain multiple variants on a single read, which demonstrates that those variants on that read come from the same chromosome. However, because the accuracy of these long reads is much lower on a per-base level compared to short shotgun reads, it is more difficult to determine whether observed differences in bases in sequencing reads at a given locus are due to an actual variation in the DNA or sequencing errors (as pictured in Figure 2.3), making both genotyping and phasing more challenging to do accurately. If variant callers can handle high error rates in the reads appropriately, variant calling programs designed for use with long reads have the potential to find variants in previously unmappable regions of the genome, and phase variants more accurately.

Even though there are existing tools for both variant calling and phasing, there are none that do both genotyping and phasing, besides WhatsHap [22] (whose authors we collaborated with as part of this work). Therefore, there is room for additional technologies to be introduced to this space. We propose a program that simultaneously does both genotyping and phasing, using an HMM to bipartition the DNA sequencing reads into their haplotypes and determine the genotype of each haplotype individually.

Implementation

In the remaining portion of this chapter, I discuss the work we have completed to implement a tool that simultaneously does both variant calling and phasing using long reads, called `MarginPhase`. Additional background relevant to the paper can be

found within the included portion of the main text.

Contributions

The following contains the full text of the MarginPhase paper, called "Haplotype-aware diplotyping from noisy long reads". I was co-first author along with Trevor Pesout and Jana Ebler, and the paper was published in *Genome Biology* in June 2019. I worked on the implementation of the MarginPhase software, including coding and testing its components. I also worked on designing and executing the experiments included in the paper. I also wrote a significant portion of the main text, and worked on figure design and production.

To give a more thorough breakdown of my individual contributions to this paper, I will give my own perspective on the work and its components. As a broad overview, we proposed using a Hidden Markov Model (HMM) to bipartition the reads into two sets, each set arising from the same chromosome, and then genotype and phase within the resulting bipartitions. This should improve the accuracy of genotyping and phasing if we are able to perform both processes with only the reads that are most likely to be in each haplotype, because it will be easier to identify sequencing errors (as opposed to not having them separated out, where errors and heterozygous sites can be hard to distinguish). In addition to improving the accuracy of these processes, this would have the added bonus that both the genotyping and phasing will be completed at the same time.

The first goal was to implement the diplotyper, thoroughly testing it along the way. Trevor Pesout and I were given a prototype of the MarginPhase program, written by our advisor Benedict Paten, that had an implementation of an Hidden Markov Model (HMM) that could bipartition a set of simulated test reads, and then genotype and phase within the read bipartitions. We extended this implementation to read in real sequencing reads, get those to work within the HMM, and output a final, phased VCF at the end. Though it sounded like a good portion of the work was done with the working prototype, the resulting engineering to make the tool usable took quite a while longer. There were many iterations of testing and development to parameterize the model to work with the real sequencing error modalities onserved in PacBio and ONT read sets.

We collaborated with Jana Ebler and Tobias Marschall to publish this work. They were simultaneously working on `WhatsHap`, which is another tool that could simultaneously genotype and phase variants using long reads. We designed experiments to compare our two tools and explore their uses. We tested the tools on a standard benchmark set (NA12878), using both PacBio and ONT reads to test the accuracy of both genotyping and phasing. We found that each tool had its own strengths (for example, WhatsHap had better genotyping accuracy using PacBio reads, and MarginPhase had better accuracy with ONT reads). Both tools allowed for the identification of many hundreds of thousands of variants outside of the regions that were previously considered to be the 'genotypable' regions of the genome, because the long reads were able to be mapped to a larger portion of the genome.

Related work on MarginPhase after publication

After the time of publication of this initial paper, WhatsHap went on to become the industry standard for phasing. (MarginPhase was largely a proof-of-concept at this point, and had some limitations involving usability.) However, the code for MarginPhase has provided a framework, now called 'Margin', upon which Trevor Pesout greatly expanded for the rest of his dissertation. MarginPolish was the next tool developed, which is a polisher to improve erroneous sequence for de novo assemblies. It was introduced in the paper upon which the next chapter of my dissertation is based, "Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes" [23]. Next, Margin was used to phase reads in a variant calling pipeline called `PEPPER-Margin-DeepVariant`, published in the paper "Haplotype-aware variant calling with PEPPER-Margin-DeepVariant enables high accuracy in nanopore long-reads" [24]. The PEPPER-Margin-DeepVariant pipeline has gone on to be used within ultrarapid clinical testing settings [25, 26], and at its fastest a sample was able to be collected, sequenced, analyzed, and diagnosed in under 8 hours.

Full Text of the MarginPhase paper

Abstract

Current genotyping approaches for single-nucleotide variations rely on short, accurate reads from second-generation sequencing devices. Presently, third-generation

sequencing platforms are rapidly becoming more widespread, yet approaches for leveraging their long but error-prone reads for genotyping are lacking. Here, we introduce a novel statistical framework for the joint inference of haplotypes and genotypes from noisy long reads, which we term diplotyping. Our technique takes full advantage of linkage information provided by long reads. We validate hundreds of thousands of candidate variants that have not yet been included in the high-confidence reference set of the Genome-in-a-Bottle effort.

Background

Reference-based genetic variant identification comprises two related processes: genotyping and phasing. Genotyping is the process of determining which genetic variants are present in an individual's genome. A genotype at a given site describes whether both chromosomal copies carry a variant allele, whether only one of them carries it, or whether the variant allele is not present at all. Phasing refers to determining an individual's haplotypes, which consist of variants that lie near each other on the same chromosome and are inherited together. To completely describe all of the genetic variation in an organism, both genotyping and phasing are needed. Together, the two processes are called diplotyping.

Many existing variant analysis pipelines are designed for short DNA sequencing reads [27, 28]. Though short reads are very accurate at a per-base level, they can suffer from being difficult to unambiguously align to the genome, especially in repet-

itive or duplicated regions [29]. The result is that millions of bases of the reference human genome are not currently reliably genotyped by short reads, primarily in multi-megabase gaps near the centromeres and short arms of chromosomes [30]. While short reads are unable to uniquely map to these regions, long reads can potentially span into or even across them. Long reads have already proven useful for read-based haplotyping, large structural variant detection, and *de novo* assembly [31, 32, 33, 34]. Here, we demonstrate the utility of long reads for more comprehensive genotyping. Due to the historically greater relative cost and higher sequencing error rates of these technologies, little attention has been given thus far to this problem. However, long-read DNA sequencing technologies are rapidly falling in price and increasing in general availability. Such technologies include single-molecule real-time (SMRT) sequencing by Pacific Biosciences (PacBio) and nanopore sequencing by Oxford Nanopore Technologies (ONT), both of which we assess here.

The genotyping problem is related to the task of inferring haplotypes from long-read sequencing data, on which a rich literature and many tools exist [8–14], including our own software WhatsHap [35, 22]. The most common formalization of haplotype reconstruction is the minimum error correction (MEC) problem. The MEC problem seeks to partition the reads by haplotype such that a minimum number of errors need to be corrected in order to make the reads from the same haplotype consistent with each other. In principle, this problem formulation could serve to infer genotypes, but in practice, the "all heterozygous" assumption is made: tools for haplotype reconstruction generally assume that a set of heterozygous positions is given as input and exclusively

work on these sites.

Despite this general lack of tools, some methods for genotyping using long reads have been proposed. Guo et al. [36] describe a method for long-read single-nucleotide variant (SNV) calling and haplotype reconstruction which identifies an exemplar read at each SNV site that best matches nearby reads overlapping the site. It then partitions reads around the site based on similarity to the exemplar at adjacent SNV sites. However, this method is not guaranteed to discover an optimal partitioning of the reads between haplotypes, and the authors report a comparatively high false discovery rate (15.7%) and false-negative rate (11.0%) for PacBio data of NA12878, which corresponds to an F1 score of only 86.6%. Additionally, two groups are presently developing learning-based variant callers which they show can be tuned to work using long, noisy reads: In a recent preprint, Luo et al. [17] describe a method which uses a convolutional neural network (CNN) to call variants from long-read data, which they report to achieve an F1 score between 94.90 and 98.52%, depending on parametrization (when training on read data from one individual and calling variants on a different individual, see Table 3 of [17]). Poplin et al. [16] present another CNN-based tool, which achieves an F1 score of 92.67% on PacBio data (according to Supplementary Table 3 of [19]). These measures appear promising; however, these methods do not systematically exploit the linkage information between variants provided by long reads. Thus, they do not leverage one of the key advantages of long reads.

For an illustration of the potential benefit of using long reads to diplotype across adjacent sites, consider Figure 2.4a. There are three SNV positions shown which

are covered by long reads. The gray sequences represent the true haplotype sequences, and reads are colored in blue and red, where the colors correspond to the haplotype which the respective read stems from: the red ones from the upper sequence, and the blue ones from the lower one. Since sequencing errors can occur, the alleles supported by the reads are not always equal to the true ones in the haplotypes shown in gray. Considering the SNVs individually, it would be reasonable to call the first one as A/C, the second one as T/G, and the third one as G/C, since the number of reads supporting each allele is the same. This leads to a wrong prediction for the second SNV. However, if we knew which haplotype each read stems from, that is, if we knew their colors, then we would know that there must be sequencing errors at the second SNV site. Since the reads stemming from the same haplotypes must support the same alleles and there are discrepancies between the haplotyped reads at this site, any genotype prediction at this locus must be treated as highly uncertain. Therefore, using haplotype information during genotyping makes it possible to detect uncertainties and potentially compute more reliable genotype predictions.

Contributions

In this paper, we show that for contemporary long read technologies, read-based phase inference can be simultaneously combined with the genotyping process for SNVs to produce accurate diplotypes and to detect variants in regions not mappable by short reads. We show that key to this inference is the detection of linkage relationships between heterozygous sites within the reads. To do this, we describe a novel algorithm

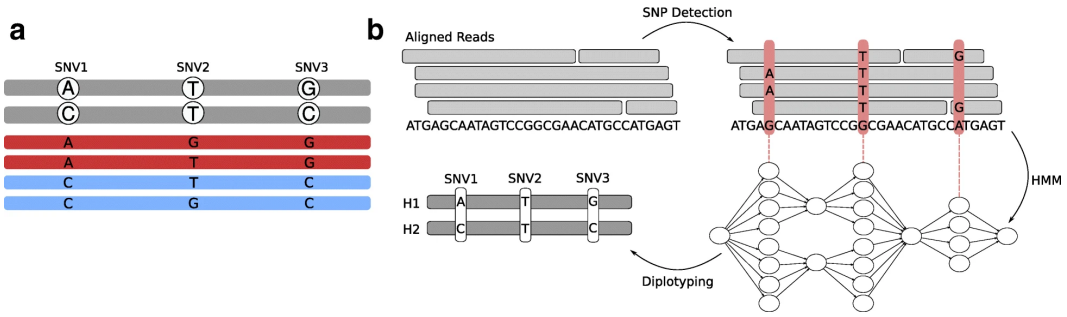

Figure 2.4: **Motivation and overview of diplotyping.** a Gray sequences illustrate the haplotypes; the reads are shown in red and blue. The red reads originate from the upper haplotype, the blue ones from the lower. Genotyping each SNV individually would lead to the conclusion that all of them are heterozygous. Using the haplotype context reveals uncertainty about the genotype of the second SNV. b Clockwise starting top left: first, sequencing reads aligned to a reference genome are given as input; second, the read alignments are used to nominate candidate variants (red vertical bars), which are characterized by the differences to the reference genome; third, a hidden Markov model (HMM) is constructed where each candidate variant gives rise to one "row" of states, representing possible ways of assigning each read to one of the two haplotypes as well as possible genotypes (see the "Methods" section for details); forth, the HMM is used to perform diplotyping, i.e., we infer genotypes of each candidate variant as well as how the alleles are assigned to haplotypes

to accurately predict diplotypes from noisy long reads that scales to deeply sequenced human genomes.

We then apply this algorithm to diplotype one individual from the 1000 Genomes Project, NA12878, using long reads from both PacBio and ONT. NA12878 has been extensively sequenced and studied, and the Genome in a Bottle Consortium has published

sets of high confidence regions and a corresponding set of highly confident variant calls inside these genomic regions [37]. We demonstrate that our method is accurate, that it can be used to confirm variants in regions of uncertainty, and that it allows for the discovery of variants in regions which are unmappable using short DNA read sequencing technologies.

Results

A unified statistical framework to infer genotypes and haplotypes

We formulated a novel statistical framework based upon hidden Markov models (HMMs) to analyze long-read sequencing data. In short, we identify potential SNV positions and use our model to efficiently evaluate the bipartitions of the reads, where each bipartition corresponds to assigning each read to one of the individual's two haplotypes. The model ensures that each read stays in the same partition across variants, and hence does not "switch haplotypes," something which is key to exploiting the inherent long range information. Based on the read support of each haplotype at each site, the model determines the likelihood of the bipartition. By using the forward-backward algorithm, we pursue "global" diplotype inference over whole chromosomes, a process that yields genotype predictions by determining the most likely genotype at each position, as well as haplotype reconstructions. In contrast to panel-based methods, like the Li-Stephens model [38], our method relies on read data instead of using knowledge of existing haplotypes. In 2.4b, we give a conceptual overview of our approach and

describe it in more detail in the "Methods" section.

Adding robustness to our analysis, we provide two independent software implementations of our model: one is made available as an extension to WhatsHap [22, 39], and the other is a from-scratch implementation called MarginPhase. While the core algorithmic ideas are the same, MarginPhase and WhatsHap differ primarily in their specialized preprocessing steps, with the former being developed to work primarily with nanopore data and the latter developed to work primarily with PacBio (although both can work with either). The MarginPhase workflow includes an alignment summation step described in the "Allele supports" section whereas WhatsHap performs a local realignment around analyzed sites explained in the "Allele detection" section.

Data preparation and evaluation

To test our methods, we used sequencing data for NA12878 from two different long-read sequencing technologies. NA12878 is a participant from the 1000 Genomes Project [28] who has been extensively sequenced and analyzed. This is the only individual for whom there is both PacBio and Nanopore sequencing reads publicly available. We used Oxford Nanopore reads from Jain et al. [33] and PacBio reads from the Genome in a Bottle Consortium [40]. Both sets of reads were aligned to GRCh38 with minimap2, a mapper designed to align error-prone long reads [41] (version 2.10, using default parameters for PacBio and Nanopore reads, respectively).

To ensure that any variants we found were not artifacts of misalignment, we filtered out the reads flagged as secondary or supplementary, as well as reads with a

mapping quality score less than 30. Genome-wide, this left approximately 12 million Nanopore reads and 35 million PacBio reads. The Nanopore reads had a median depth of 37 × and median length of 5950 bp, including a set of ultra-long reads with lengths up to 900 kb. The PacBio reads had a median depth of 46 × and median length of 2600 bp.

To validate the performance of our methods, we used callsets from Genome in a Bottle's (GIAB) benchmark small variant calls v3.3.2 [37]. First, we compared against GIAB's set of high confidence calls, generated by a consensus algorithm spanning multiple sequencing technologies and variant calling programs. The high confidence regions associated with this callset exclude structural variants, centromeres, and heterochromatin. We used this to show our method's accuracy in well-understood and easy-to-map regions of the genome.

We also analyzed our results compared to two more expansive callsets, which cover a larger fraction of the genome, that were used in the construction of GIAB's high confidence variants, one made by GATK HaplotypeCaller v3.5 (GATK/HC, [27]) and the other by Freebayes 0.9.20 [14], both generated from a 300× PCR-free Illumina sequencing run [37].

Evaluation statistics

We compute the precision and recall of our callsets using the tool vcfeval from Real Time Genomics [42] (version 3.9) in order to analyze our algorithm's accuracy of variant detection between our query callsets and a baseline truth set of variants. All

variants that identically match between the truth and query callsets (meaning they share the same genomic position, alleles, and genotype) are considered true positive calls. Calls that do not match any variants in the truth callset are false negatives and truth callset variants that are not matched in our callset are false positives.

In order to evaluate the ability of our algorithm to genotype a provided set of variant positions, we compute the genotype concordance. Here, we take all correctly identified variant sites (correct genomic position and alleles), compare the genotype predictions (homozygous or heterozygous) made by our method to the corresponding truth set genotypes, and compute the fraction of correct genotype predictions. This enables us to analyze how well the genotyping component of our model performs regardless of errors arising from wrongly called variant sites in the detection stage of the algorithm.

We evaluate the phasing results by computing the switch error rate between the haplotypes our algorithms predict and the truth set haplotypes. We take all variants into account that were correctly genotyped as heterozygous in both our callset and the truth set. Switch errors are calculated by counting the number of times a jump from one haplotype to the other is necessary within a phased block in order to reconstruct the true haplotype sequence [22].

We restrict all analysis to SNVs, not including any short insertions or deletions. This is due to the error profile of both PacBio and Nanopore long reads, for which erroneous insertions and deletions are the most common type of sequencing error by far, particularly in homopolymers [27, 28].

Comparison to short read variant callers

We explored the suitability of the current state-of-the-art callers for short reads to process long-read data (using default settings) but were unsuccessful. The absence of base qualities in the PacBio data prevented any calling; for Nanopore data, FreeBayes was prohibitively slow and neither Platypus nor GATK/HC produced any calls.

Long read coverage

We determined the regions where long and short reads can be reliably mapped to the human genome for the purpose of variant calling, aiming to understand if long reads could potentially make new regions accessible. In Fig. 2, various coverage metrics for short and long reads are plotted against different genomic features, including those known for being repetitive or duplicated. These metrics are described below.

The callsets on the Illumina data made by GATK/HC and FreeBayes come with two BED files describing where calls were made with some confidence. The first, described in Fig. 2 as Short Read Mappable, was generated using GATK CallableLoci v3.5 and includes regions where there is (a) at least a read depth of 20 and (b) at most a depth of twice the median depth, only including reads with mapping quality of at least 20. This definition of callable only considers read mappings.

The second, described as GATK Callable, was generated from the GVCF output from GATK/HC by excluding the areas with genotype quality less than 60. This is a more sophisticated definition of callable as it reflects the effects of homopolymers and tandem repeats. We use these two BED files in our analysis of how short and long

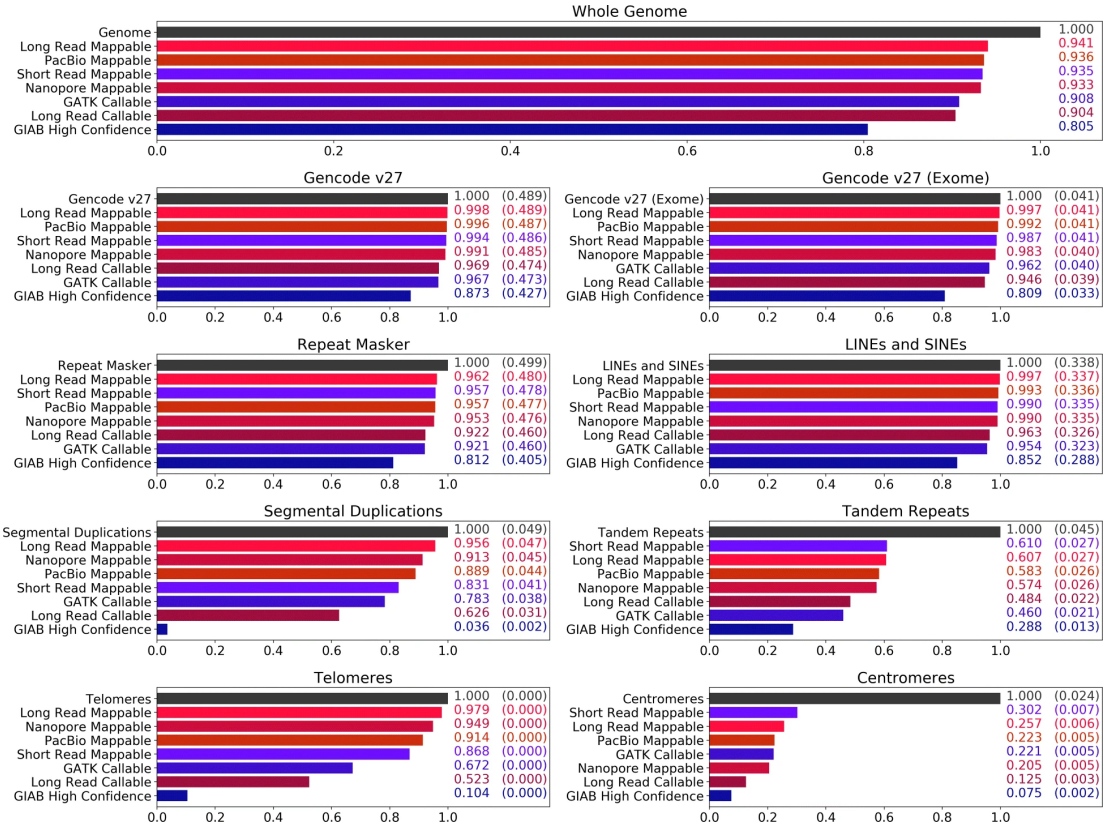

Figure 2.5: **Reach of short read and long read technologies.** The callable and mappable regions for NA12878 spanning various repetitive or duplicated sequences on GRCh38 are shown. Feature locations are determined based on BED tracks downloaded from the UCSC Genome Browser [48]. Other than the Gencode regions [49, 50], all features are subsets of the Repeat Masker [51] track. Four coverage statistics for long reads (shades of red) and three for short reads (shades of blue) are shown. The labels "PacBio Mappable" and "Nanopore Mappable" describe areas where at least one primary read with GQ ≥ 30 has mapped, and "Long Read Mappable" describes where this is true for at least one of the long read technologies. "Long Read Callable" describes areas where both read technologies have coverage of at least 20 and less than twice the median coverage. "GIAB High Confidence," "GATK Callable," and "Short Read Mappable" are the regions associated with the evaluation callsets. For the feature-specific plots, the numbers on the right detail coverage over the feature and coverage over the whole genome (parenthesized)

reads map differently in various areas of the genome.

For long reads, we show four coverage statistics. The entries marked as "mappable" describe the areas where there is at least one high-quality long-read mapping (PacBio Mappable, Nanopore Mappable, and Long Read Mappable for regions where

at least one of the sequencing technologies mapped). The Long Read Callable entries cover the regions where our methods should be able to call variants due to having a sufficient depth of read coverage. In these regions, both sequencing technologies had a minimum read depth of 20 and a maximum of twice the median depth (this is similar to the GATK CallableLoci metric, although made from BAMs with significantly less read depth).

Figure 2 shows that in almost all cases, long reads map to a higher fraction of the genome than short reads map to. For example, nearly half a percent of the whole genome is mappable by long reads but not short reads. Long reads also map to 1% more of the exome, and 13% more of segmental duplications. Centromeres and tandem repeats are outliers to this generalization, where neither PacBio nor Nanopore long reads cover appreciably more than Illumina short reads.

Comparison against high confidence truth set

To validate our method, we first analyzed the SNV detection and genotyping performance of our algorithm using the GIAB high confidence callset as a benchmark. All variants reported in these statistics fall within both the GIAB high confidence regions and regions with a read depth between 20 and twice the median depth.

Variant detection

Figure 3 (top) shows precision and recall of WhatsHap run on PacBio data and MarginPhase on Oxford Nanopore data, which gives the best performance for these two

data types (see Additional file 1: Figure S1 for the results for WhatsHap on ONT and MarginPhase on PacBio). On PacBio reads, WhatsHap achieves a precision of 97.9% and recall of 96.3%. On Nanopore reads, MarginPhase achieves a precision of 76.9% and a recall of 80.9%. We further stratify the performance of our methods based on the variant type. For homozygous variants, WhatsHap on PacBio data has a precision of 98.3% and a recall of 99.3%, MarginPhase on Nanopore data has a precision of 99.3% and a recall of 84.5%. For heterozygous variants, WhatsHap on PacBio data has a precision of 96.8% and a recall of 93.8%; MarginPhase on Nanopore data has a precision of 66.5% and a recall of 78.6%. The high error rate of long reads contributes to the discrepancy in the performance between homozygous and heterozygous variant detection, making it more difficult to distinguish the read errors from the alternate allele for heterozygous variants. In Section 5 of Additional file 1, we further discuss the precision and recall as a function of read depth, and we report more performance based on variant type in Section 6.

Long reads have the ability to access regions of the genome inaccessible to short reads ("Long read coverage" section). To explore the potential of our approach to contribute to extending gold standard sets, such as the one produced by the GIAB effort, we produced a combined set of variants which occur in both the calls made by WhatsHap on the PacBio reads and MarginPhase on the Nanopore data, where both tools report the same genotype. This improves the precision inside the GIAB high confidence regions to 99.7% with a recall of 78.7%. In further analysis, we refer to this combined variant set as Long Read Variants. It reflects a high precision subset

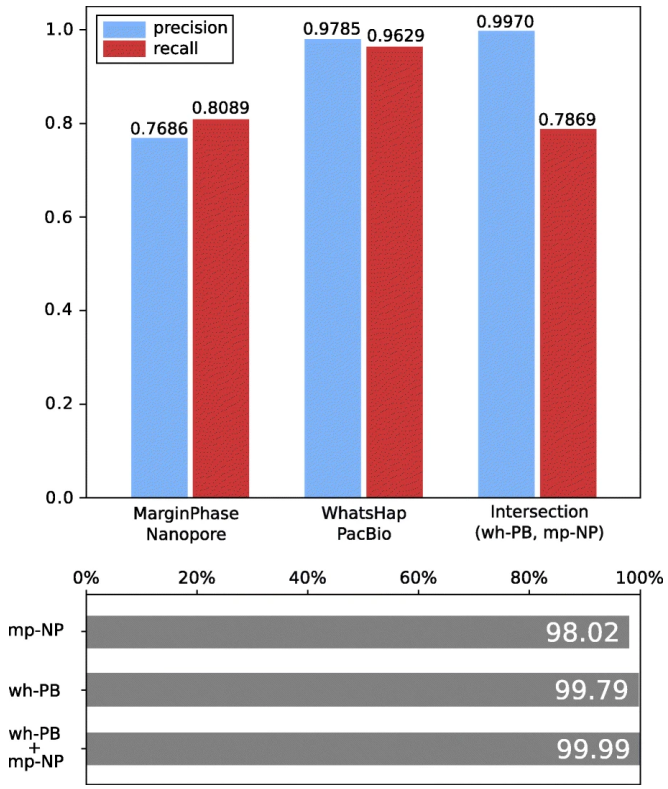

Figure 2.6: **Precision and recall of MarginPhase on Nanopore and WhatsHap on PacBio datasets in GIAB high confidence regions.**Genotype concordance (bottom) (wrt. GIAB high confidence calls) of MarginPhase (mp, top) on Nanopore and WhatsHap (wh, middle) on PacBio (PB). Furthermore, genotype concordance for the intersection of the calls made by WhatsHap on the PacBio and MarginPhase on the Nanopore reads is shown (bottom)

of variants validated independently by both sequencing technologies. While data from both technologies are usually not available for the same sample in routine settings, such a call set can be valuable for curating variants on well-studied individuals such as NA12878.

Genotyping

In order to further analyze the quality of the genotype predictions of our methods (heterozygous or homozygous), we computed the genotype concordance (defined in the "Data preparation and evaluation" section) of our callsets with respect to the GIAB ground truth inside of the high confidence regions. Figure 3 (bottom) shows the results. On the PacBio data, WhatsHap obtains a genotype concordance of 99.79. On the Nanopore data, MarginPhase obtains a genotype concordance of 98.02. Considering the intersection of the WhatsHap calls on PacBio, and MarginPhase calls on Nanopore data (i.e., the Long Read Variants set), we obtain a genotype concordance of 99.99%. We detail the genotype performances for different thresholds on the genotype quality scores that our methods report for each variant call (Additional file 1: Section 7).

Phasing

In addition to genotyping variants, MarginPhase and WhatsHap can also phase them. We evaluated the results of both methods by computing switch error rates (defined in the "Data preparation and evaluation" section) inside the GIAB high-confidence regions for correctly located and genotyped GIAB truth set variants. We computed the switch error rate of MarginPhase on Nanopore and WhatsHap on PacBio reads. For both datasets, we achieved a low switch error rate of 0.17%. In Additional file 1: Table S1, corresponding per-chromosome switch error rates are given.

Cutting and downsampling reads

Our genotyping model incorporates haplotype information into the genotyping process by using the property that long sequencing reads can cover multiple variant positions. Therefore, one would expect the genotyping results to improve as the length of the provided sequencing reads increases.

In order to examine how the genotyping performance depends on the length of the sequencing reads and the coverage of the data, the following experiment was performed using the WhatsHap implementation. The data was downsampled to average coverages 10×,20×,25×, and 30×. All SNVs inside of the high confidence regions in the GIAB truth set were re-genotyped from each of the resulting downsampled read sets, as well as from the full coverage data sets. Two versions of the genotyping algorithm were considered. First, the full-length reads as given in the BAM files were provided to WhatsHap. Second, in an additional step prior to genotyping, the aligned sequencing reads were cut into shorter pieces such that each resulting fragment covered at most two variants. Additionally, we cut the reads into fragments covering only one variant position. The genotyping performances of these genotyping procedures were finally compared by determining the amount of incorrectly genotyped variants.

Figure 4 shows the results of this experiment for the PacBio data. The genotyping error increases as the length of reads decreases. Especially at lower coverages, the genotyping algorithm benefits from using the full length reads, which leads to much lower genotyping errors compared to using the shorter reads that lack information of

neighboring variants. For the Nanopore reads, the results were very similar (Additional file 1: Figure S2). In general, the experiment demonstrates that incorporating haplotype information gained from long reads does indeed improve the genotyping performance. This is especially the case at low coverages, since here, the impact of sequencing errors on the genotyping process is much higher. Computing genotypes based on bipartitions of reads that represent possible haplotypes of the individual helps to reduce the number of genotyping errors, because it makes it easier to detect sequencing errors in the given reads.

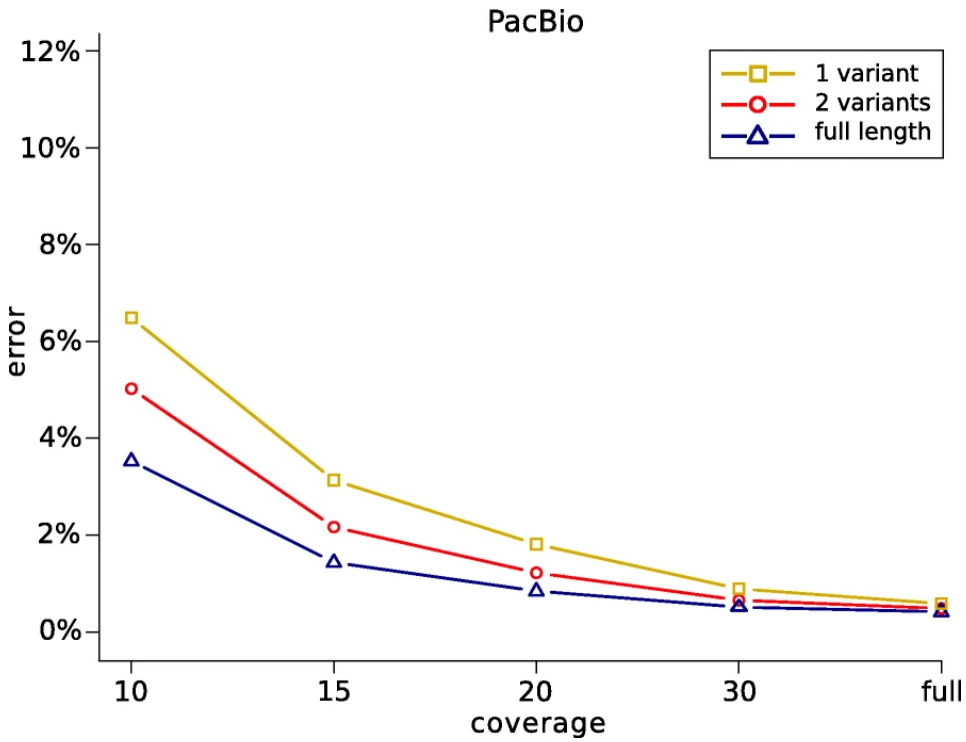

Figure 2.7: **Genotyping errors (with respect to GIAB calls) as a function of coverage.** The full length reads were used for genotyping (blue), and additionally, reads were cut such as to cover at most two variants (red) and one variant (yellow)

Callset consensus analysis

Call sets based on long reads might contribute to improving benchmark sets such as the GIAB truth set. We analyze a call set created by taking the intersection of the variants called by WhatsHap on PacBio reads and MarginPhase on Nanopore reads, which leaves variants that were called identically between the two sets. In 2.8, we further dissect the relation of this intersection callset, which we call Long Read Variants, to the GIAB truth set, as well as its relation to the callsets from GATK HaplotypeCaller and FreeBayes, which both contributed to the GIAB truth set.

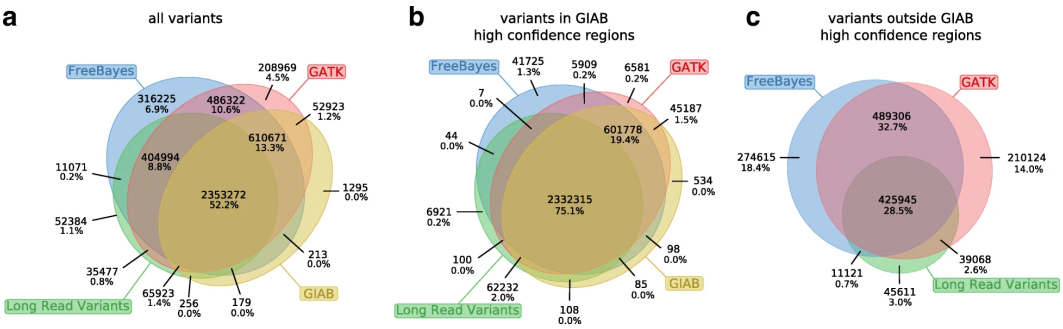

Figure 2.8: **Confirming short-read variants.** We examine all distinct variants found by our method, GIAB high confidence, GATK/HC, and FreeBayes. Raw variant counts appear on top of each section, and the percentage of total variants is shown at the bottom. a All variants. b Variants in GIAB high-confidence regions. c Variants outside GIAB high-confidence regions

2.8a reveals that 404,994 variants in our Long Read Variants callset were called by both the GATK Haplotype Caller and FreeBayes, yet are not in the GIAB truth set. To gather additional support for the quality of these calls, we consider two established quality metrics: the transition/transversion ratio (Ti/Tv) and the heterozygous/non-ref homozygous ratio (Het/Hom) [29]. The Ti/Tv ratio of these variants is 2.09, and the Het/Hom ratio is 1.31. These ratios are comparable to those of the GIAB truth set, which are 2.10 and 1.55, respectively. An examination of the Platinum Genomes benchmark set [30], an alternative to GIAB, reveals 78,493 such long-read validated variants outside of their existing truth set.

We hypothesized that a callset based on long reads is particularly valuable in the regions that were previously difficult to characterize. To investigate this, we separately examined the intersections of our Long Read Variants callset with the two short-read callsets both inside the GIAB high confidence regions and outside of them, see 2.8b and 2.8bc, respectively. These Venn diagrams clearly indicate that the concordance of GATK and FreeBayes was indeed substantially higher in high confidence regions than outside. An elevated false-positive rate of the shortread callers outside the high confidence regions is a plausible explanation for this observation. Interestingly, the fraction of calls concordant between FreeBayes and GATK for which we gather additional support is considerably lower outside the high confidence regions. This is again compatible with an increased number of false positives in the short-read callsets, but we emphasize that these statistics should be interpreted with care in the absence of a reliable truth set for these regions.

Candidate novel variants

To demonstrate that our method allows for variant calling on more regions of the genome than short-read variant calling pipelines, we have identified 15,498 variants which lie outside of the Short Read Mappable area, but inside the Long Read Callable regions. These variants therefore fall within the regions in which there is a sequencing depth of at least 10 and not more than 2 times the median depth for both long-read sequencing technologies, yet the regions are unmappable by short reads. We determined that 4.43 Mb of the genome are only mappable by long reads in this way.

2.1 provides the counts of all variants found in each of the regions from Fig. 2, as well as the counts for candidate novel variants, among the different types of genomic features described in "Long read coverage"section. Over two thirds of the candidate variants occurred in the repetitive or duplicated regions described in the UCSC Genome Browser's repeatMasker track. The transition/transversion ratio (Ti/Tv) of NA12878's 15,498 candidate variants is 1.64, and the heterozygous/homozygous ratio (Het/Hom) of these variants is 0.31. Given that we observe 1 candidate variant in every 325 haplotype bases of the 4.43 Mb of the genome only mappable by long reads, compared to 1 variant in every 1151 haplotype bases in the GIAB truth set on the whole genome, these candidate variants exhibit a 3.6× increase in the haplotype variation rate.

Runtimes

Whole-genome variant detection using WhatsHap took 166 CPU hours on PacBio reads, of which genotyping took 44 h. Per chromosome, a maximum of 4.2 GB

of memory was required for genotyping, and additionally, at most 2.6 GB was needed for phasing. The MarginPhase implementation took 1550 CPU hours on ONT data, broken down into 330 h for diplotyping and 1220 h for read realignment (described in the "Allele supports" section). The MarginPhase workflow breaks the genome into 2-Mb overlapping windows, and on each of these windows, MarginPhase required on average 22.6 GB of memory, and a maximum of 30.2 GB.

We found that the time-consuming realignment step significantly improved the quality of the ONT results and attribute this as the major cause of the difference in runtimes. Furthermore, the methods employed to the find candidate sites differ between the implementations. WhatsHap performs genotyping and phasing in two steps, whereas MarginPhase handles them simultaneously after filtering out the sites that are likely homozygous (in the case of ONT data, this is between 98 and 99% of sites). The filtration heuristic used during our evaluation resulted in MarginPhase analyzing roughly 10× the number of sites than WhatsHap, increasing the runtime and

Table 2.1: Distribution of candidate novel variants across different regions of interest. All variants refers to the variants in the Long Read Variants set, and Novel Variant Candidates are those described in Section 3.6.

	All Variants	**Novel Variant Candidates**
Total	2,913,942	15,498
Gencode v27 (ALL)	1,363,064	5,594
Gencode v27 exome	86,357	538
Repeat Masker	1,583,684	10,677
LINEs	690,859	5,161
SINEs	421,340	1,432
Segmental Duplications	157,341	5,683
Tandem Repeats	96,871	5,437
Centromeres	18,644	2,031
Telomeres	295	14

memory usage.

Discussion

We introduce a novel statistical model to unify the inference of genotypes and haplotypes from long (but noisy) third-generation sequencing reads, paving the way for genotyping at intermediate coverage levels. We emphasize that our method operates at coverage levels that preclude the possibility of performing a de novo genome assembly, which, until now, was the most common use of long-read data. Furthermore, we note that unlike the approaches using a haplotype reference panel of a population for statistical phasing and/or imputation [31], our approach only uses sequencing data from the individual; hence, its performance does not rely on the allele frequency within a population.

Our method is based on a hidden Markov model that partitions long reads into haplotypes, which we found to improve the quality of variant calling. This is evidenced by our experiment in cutting and downsampling reads, where reducing the number of variants spanned by any given read leads to decreased performance at all levels of read coverage. Therefore, our method is able to translate the increased read lengths of third generation platforms into increased genotyping performance for these noisy long reads.

Our analysis of the methods against a high confidence truth set in high confidence regions shows false discovery rates (corresponding to one minus precision) between 3 and 6% for PacBio and between 24 and 29% for Nanopore. However, when considering

a conservative set of variants confirmed by both long read technologies, the false discovery rate drops to around 0.3%, comparable with contemporary short-read methods in these regions.

In analyzing the area of the genome with high-quality long-read mappings, we found roughly a half a percent of the genome (approximately 15 Mb) that is mappable by long reads but not by short reads. This includes 1% of the human exome, as well as over 10% of segmental duplications. Even though some of these areas have low read counts in our experimental data, the fact that they have high-quality mappings means that they should be accessible with sufficient sequencing. We note that this is not the case for centromeric regions, where Illumina reads were able to map over twice as much as we found in our PacBio data. This may be a result of the low quality in long reads preventing them from uniquely mapping to these areas with an appreciable level of certainty.

We demonstrate that our callset has expected biological features, by showing that over our entire set of called variants, the Ti/Tv and Het/Hom ratios were similar to those reported by the truth set. The Ti/Tv ratio of 2.18 is slightly above the 2.10 reported in the GIAB callset, and the Het/Hom ratio of 1.36 is slightly lower than the 1.55 found in the GIAB variants. In the 15,498 novel variant candidates produced by our method in regions unmappable by short reads, the Ti/Tv ratio of 1.64 is slightly lower than that of the truth set. This is not unexpected as gene-poor regions such as these tend to have more transversions away from C:G pairs [32]. We also observe that the Het/Hom ratio dropped to 0.31, which could be due to the systematic biases in

our callset or in the reference genome. The rate of variation in these regions was also notably different than in the high confidence regions, where we find three variants per thousand haplotype bases (3.6× the rate in high confidence regions). A previous study analyzing NA12878 [33] also found an elevated variation rate in the regions where it is challenging to call variants, such as low-complexity regions and segmental duplications. The study furthermore found clusters of variants in these regions, which we also observe.

The high precision of our set containing the intersection of variants called on Nanopore reads and variants called on PacBio reads makes it useful as strong evidence for confirming existing variant calls. As shown in the read coverage analysis, in both the GIAB and Platinum Genomes efforts many regions could not be called with high confidence. In the regions excluded from GIAB, we found around 400,000 variants using both Nanopore and PacBio reads with our methods, which were additionally confirmed by 2 other variant callers, FreeBayes and GATK/HC, on Illumina reads. Given the extensive support of these variants from multiple sequencing technologies and variant callers, these 400,000 variants are good candidates for addition to the GIAB truth set. Expansion of benchmark sets to harder-to-genotype regions of the human genome is generally important for the development of more comprehensive genotyping methods, and we plan to work with these efforts to use our results.

Conclusions

Variant calling with long reads is difficult because they are lossy and error-prone, but the diploid nature of the human genome provides a means to partition reads to lessen this effect. We exploit the fact that reads spanning heterozygous sites must share the same haplotype to differentiate read errors from true variants. We provide two implementations of this method in two long-read variant callers, and while both implementations can be run on either sequencing technology, we currently recommend that MarginPhase is used on ONT data and that WhatsHap is used on PacBio data.

One way we anticipate improvement to our method is by incorporating methylation data. Hidden Markov models have been used to produce methylation data for ONT reads using the underlying signal information [34, 35]. As shown by the read-cutting experiment, the amount of heterozygous variants spanned by each read improves our method's accuracy. We predict that the inclusion of methylation into the nucleotide alphabet will increase the amount of observable heterozygosity and therefore further improve our ability to call variants. Work has begun to include methylation probabilities into our method.

The long-read genotyping work done by Luo et al. [17] using CNNs does not account for haplotype information. Partitioning reads into haplotypes as a preprocessing step (such as our method does) may improve the CNN's performance; we think this is an interesting avenue of exploration.

Further, our method is likely to prove useful for future combined diplotyp-

ing algorithms when both genotype and phasing is required, for example, as may be used when constructing phased diploid de novo assemblies [36, 37] or in future hybrid long/short-read diplotyping approaches. Therefore, we envision the statistical model introduced here to become a standard tool for addressing a broad range of challenges that come with long-read sequencing of diploid organisms.

Methods

We describe a probabilistic model for diplotype and genotype inference, and in this paper use it to find maximum posterior probability genotypes. The approach builds upon the WhatsHap approach [35], but incorporates a full probabilistic allele inference model into the problem. It has similarities to that proposed by [43], but we here frame the problem using Hidden Markov Models (HMMs).

Alignment matrix

Let $\mathbf{M}$ be an alignment matrix whose rows represent sequencing *reads* and whose columns represent genetic *sites*. Let m be the number of rows, let n be the number of columns, and let $\mathbf{M}_{i,j}$ be the jth element in the ith row. In each column let $\Sigma_j \subset \Sigma$ represent the set of possible *alleles* such that $\mathbf{M}_{i,j} \in \Sigma_j \cup \{-\}$, the "$-$" gap symbol representing a site at which the read provides no information. We assume no row or column is composed only of gap symbols, an uninteresting includegraphicsedge case. An example alignment matrix is shown in Figure 2.9. Throughout the following we will be informal and refer to a row i or column j, being clear from the context

whether we are referring to the row or column itself or the coordinate.

	1	2	3	4	5
1	A	G	T	-	-
2	A	G	T	-	-
3	-	C	-	G	-
4	-	C	T	G	-
5	-	-	T	C	T
6	-	-	T	C	T

Figure 2.9: **Alignment matrix.** Here, the alphabet of possible alleles is the set of DNA nucleotides, i.e., $\Sigma = \{A, C, G, T\}$

Genotype inference problem overview

A diplotype $H = (H^1, H^2)$ is a pair of haplotype (segments); a *haplotype (segment)* $H^k = H^k_1, H^k_2, \ldots, H^k_n$ is a sequence of length n whose elements represents alleles such that $H^k_j \in \Sigma_j$. Let $B = (B^1, B^2)$ be a bipartition of the rows of $\mathbf{M}$ into two parts (sets): B^1, the first part, and B^2, the second part. We use bipartitions to represent which haplotypes the reads came from, of the two in a genome. By convention we assume that the first part of B are the reads arising from H^1 and the second part of B are the reads arising from H^2.

The problem we analyze is based upon a probabilistic model that essentially represents the (Weighted) Minimum Error Correction (MEC) problem [44, 45], while modeling the evolutionary relationship between the two haplotypes and so imposing a cost on bipartitions that create differences between the inferred haplotypes.

For a bipartition B, and making an i.i.d. assumption between sites in the reads:

$$P(H|B,\mathbf{M}) = \prod_{j=1}^{n} \sum_{Z_j \in \Sigma_j} P(H_j^1|B^1, Z_j) P(H_j^2|B^2, Z_j) P(Z_j)$$

Here $P(Z_j)$ is the prior probability of the ancestral allele Z_j of the two haplotypes at column j, by default we can use a simple flat distribution over ancestral alleles (but see below). The posterior probability $P(H_j^k|B^k, Z_j) =$

$$\frac{P(H_j^k|Z_j) \prod_{\{i \in B^k : \mathbf{M}_{i,j} \neq -\}} P(\mathbf{M}_{i,j}|H_j^k)}{\sum_{Y_j \in \Sigma_j} P(Y_j|Z_j) \prod_{\{i \in B^k : \mathbf{M}_{i,j} \neq -\}} P(\mathbf{M}_{i,j}|Y_j)}$$

for $k \in \{1, 2\}$, where the probability $P(H_j^k|Z_j)$ is the probability of the haplotype allele H_j^k given the ancestral allele Z_j. For this we can use a continuous time Markov model for allele substitutions, such as Jukes-Cantor [46], or some more sophisticated model that factors the similarities between alleles (see below). Similarly, $P(\mathbf{M}_{i,j}|H_j^k)$ is the probability of observing allele $\mathbf{M}_{i,j}$ in a read given the haplotype allele H_j^k.

The genotype inference problem we consider is finding for each site:

$$\arg\max_{(H_j^1, H_j^2)} P(H_j^1, H_j^2|\mathbf{M}) = \arg\max_{(H_j^1, H_j^2)} \sum_B P(H_j^1, H_j^2|B, \mathbf{M})$$

i.e. finding the genotype (H_j^1, H_j^2) with maximum posterior probability for a generative model of the reads embedded in $\mathbf{M}$.

A graphical representation of read partitions

For a column j in $\mathbf{M}$, a row i is *active* if the first non-gap symbol in row i occurs at or before column j and the last non-gap symbol in row i occurs at or after column j. Let A_j be the set of active rows of column j. For a column j a row i is *terminal* if its last non-gap symbol occurs at column j or $j = n$. Let A'_j be the set of active, non-terminal rows of column j.

Let $B_j = (B_j^1, B_j^2)$ be a bipartition of A_j into a first part B_j^1 and a second part B_j^2. Let $\mathbf{B_j}$ be the set of all possible such bipartitions of the active rows of j. Similarly, let $C_j = (C_j^1, C_j^2)$ be a bipartition of A'_j, and $\mathbf{C_j}$ be the set of all possible such bipartitions of the active, non-terminal rows of j.

For two bipartitions $B = (B^1, B^2)$ and $C = (C^1, C^2)$, B is *compatible* with C if the subset of B^1 in $C^1 \cup C^2$ is a subset of C^1, and, similarly, the subset of B^2 in $C^1 \cup C^2$ is a subset of C^2. Note this definition is symmetric and reflexive, although not transitive.

Let $G = (V_G, E_G)$ be a directed graph. The vertices V_G are the set of bipartitions of both the active rows and the active, non-terminal rows for all columns of $\mathbf{M}$ and a special *start* and *end* vertex, i.e. $V_G = \{start, end\} \cup (\bigcup_j \mathbf{B_j} \cup \mathbf{C_j})$. The edges E_G are a subset of compatibility relationships, such that (1) for all j there is an edge $(B_j \in \mathbf{B_j}, C_i \in \mathbf{C_j})$ if B_j is compatible with C_j, (2) for all $0 < j < n$ there is an edge $(C_j \in \mathbf{C_j}, B_{j+1} \in \mathbf{B_{j+1}})$ if C_j is compatible with B_{j+1}, (3) there is an edge from the start vertex to each member of $\mathbf{B_1}$, and (4) there is an edge from each member of $\mathbf{B_n}$

to the end vertex (Note that $\mathbf{C_n}$ is empty and so contributes no vertices to G). Figure 2.10 shows an example graph.

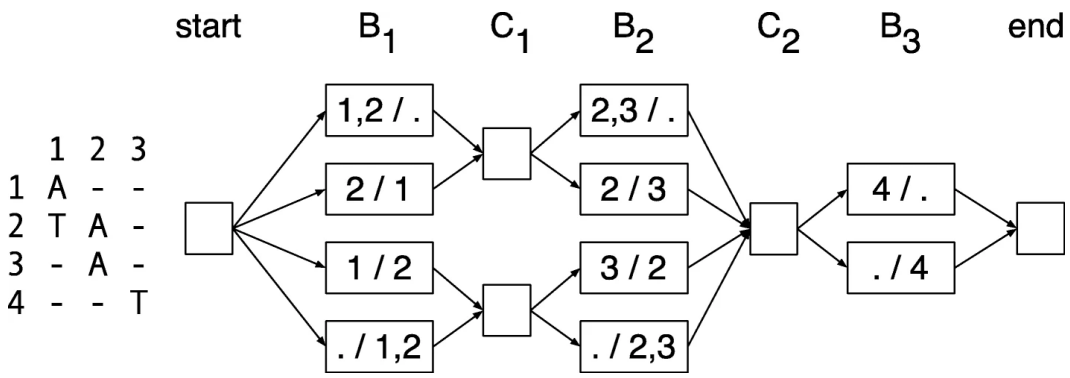

Figure 2.10: **Example graph.** Left: An alignment matrix. Right: The corresponding directed graph representing the bipartitions of active rows and active non-terminal rows, where the labels of the nodes indicate the partitions, e.g. '1,2 / .' is shorthand for $A = (\{1, 2\}, \{\})$.)

The graph G has a large degree of symmetry and the following properties are easily verified:

- For all j and all $B_j \in \mathbf{B_j}$, the indegree and outdegree of B_j is 1.
- For all j the indegree of all members of $\mathbf{C_j}$ is equal.
- Similarly, for all j the outdegree of all members of $\mathbf{C_j}$ is equal.

Let the *maximum coverage*, denoted $maxCov$, be the maximum cardinality of a set A_j over all j. By definition, $maxCov \leq m$. Using the above properties it is easily verified that: (1) the cardinality of G (number of vertices) is bounded by this maximum coverage, being less than or equal to $2 + (2n-1)2^{maxCov}$, and (2) the size of G (number of edges) is at most $2n2^{maxCov}$.

Let a directed path from the start vertex to the end vertex be called a *diploid path*, $D = (D_1 = start, D_2, \ldots, D_{2n+1} = end)$. The graph is naturally organized by the columns of $\mathbf{M}$, so that $D_{2j} = (B_j^1, B_j^2) \in \mathbf{B_j}$ and $D_{2j+1} = (C_{j+1}^1, C_{j+1}^2) \in \mathbf{C_j}$ for all $0 < j \leq n$. Let $B_D = (B_D^1, B_D^2)$ denote a pair of sets, where B_D^1 is the union of the first parts of the vertices of $D_2, \ldots, D_{2n+1}$ and, similarly, B_D^2 is the union of second parts of the vertices of $D_2, \ldots, D_{2n+1}$.

B_D^1 and B_D^2 are disjoint because otherwise there must exist a pair of vertices within D that are incompatible, which is easily verified to be impossible. Further, because D visits a vertex for every column of $\mathbf{M}$, it follows that the sum of the cardinalities of these two sets is m. B_D is therefore a bipartition of the rows of $\mathbf{M}$ which we call a *diploid path bipartition.*

Lemma 2.0.1. *The set of diploid path bipartitions is the set of bipartitions of the rows of* $\mathbf{M}$ *and each diploid path defines a unique diploid path bipartition.*

Proof. We first prove that each diploid path defines a unique bipartition of the rows of $\mathbf{M}$. For each column j of $\mathbf{M}$, each vertex $B_j \in \mathbf{B_j}$ is a different bipartition of the same set of active rows. B_j is by definition compatible with a diploid path bipartition

of a diploid path that contains it, and incompatible with every other member of $\mathbf{B_j}$. It follows that for each column j two diploid paths with the same diploid path bipartition must visit the same node in $\mathbf{B_j}$, and, by identical logic, the same node in $\mathbf{C_j}$, but then two such diploid paths are therefore equal.

There are 2^m partitions of the rows of $\mathbf{M}$. It remains to prove that there are 2^m diploid paths. By the structure of the graph, the set of diploid paths can be enumerated backwards by traversing right-to-left from the end vertex by depth-first search and exploring each incoming edge for all encountered nodes. As stated previously, the only vertices with indegree greater than one are for all j the members of $\mathbf{C_j}$, and each member of $\mathbf{C_j}$ has the same indegree. For all j the indegree of C_j is clearly $2^{|C_j|-|B_j|}$: two to the power of the number of number of active, terminal rows at column j. The number of possible paths must therefore be $\prod_{j=1}^{n} 2^{|C_j|-|B_j|}$. As each row is active and terminal in exactly one column, we obtain $m = \sum_j |C_j| - |B_j|$ and therefore:

$$2^m = \prod_{j=1}^{n} 2^{|C_j|-|B_j|}$$

. □

A hidden Markov model for genotype and diplotype inference

In order to infer diplotypes, we define a Hidden Markov Model which is based on G, but additionally represents all possible genotypes at each genomic site (i.e. in each B column). To this end, we define the set of states $\mathbf{B_j} \times \Sigma_j \times \Sigma_j$, which contains a

state for each bipartition of the active rows at position j and all possible assignments of alleles in Σ_j to the two partitions. Additionally, the HMM contains a hidden state for each bipartition in $\mathbf{C_j}$, exactly as defined for G above. Transitions between states are defined by the compatibility relationships of the corresponding bipartitions as before. This HMM construction is illustrated in Figure 2.11.

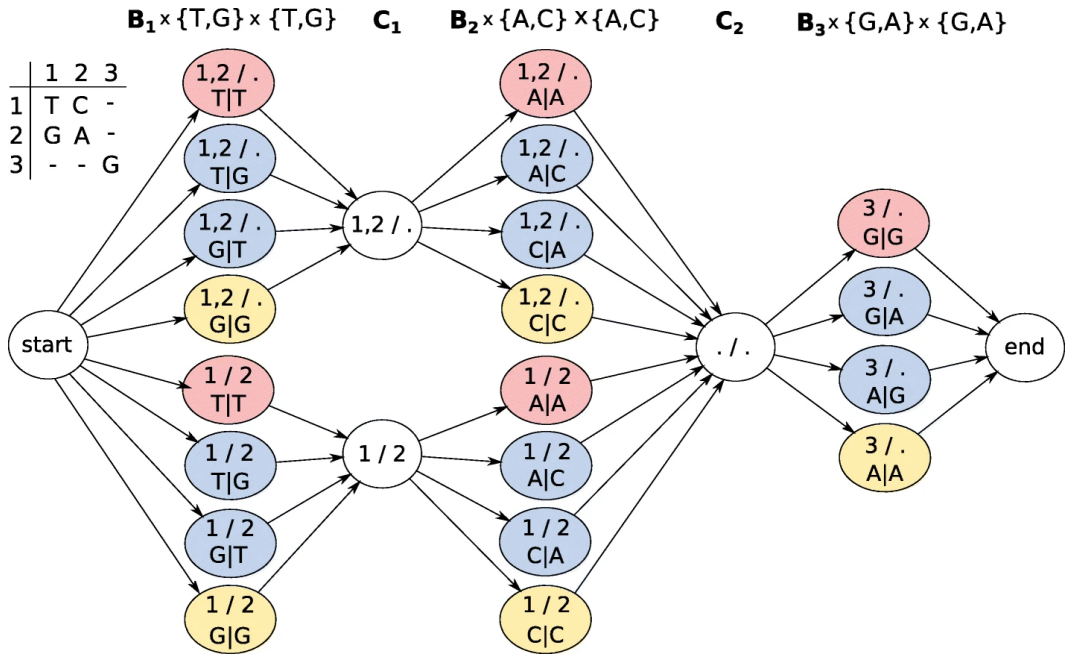

Figure 2.11: **Genotyping HMM.** Colored states correspond to bipartitions of reads and allele assignments at that position. States in C1 and C2 correspond to bipartitions of reads covering positions 1 and 2 or 2 and 3, respectively. In order to compute genotype likelihoods after running the forward-backward algorithm, states of the same color have to be summed up in each column

For all j and all $C_j \in \mathbf{C_j}$ each outgoing edge has transition probability $P(a_1, a_2) = \sum_{Z_j} P(a_1|Z_j)P(a_2|Z_j)P(Z_j)$, where $(B_j, a_1, a_2) \in \mathbf{B_j} \times \Sigma_j \times \Sigma_j$ is the state being transitioned to. Similarly, each outgoing edge of the start node has transition probability

$P(a_1, a_2)$. The outdegree of all remaining nodes is 1, so these edges have transition probability 1.

The start node, the end node, and members of $\mathbf{C_j}$ for all j are silent states, and hence do not emit symbols. For all j, members of $\mathbf{B_j} \times \Sigma_j \times \Sigma_j$ output the entries in the j-th column of $\mathbf{M}$ that are different from "–". We assume every matrix entry to be associated with an error probability, which we can compute from $P(\mathbf{M}_{ij}|H_j^k)$ defined previously. Based on this, the probability of observing a specific output column of $\mathbf{M}$ can be easily calculated.

Computing genotype likelihoods

The goal is to compute genotype likelihoods for the possible genotypes for each variant position using the HMM defined above. Performing the forward-backward algorithm returns forward and backward probabilities of all hidden states. Using those, the posterior distribution of a state $(B, a_1, a_2) \in \mathbf{B_j} \times \Sigma_j \times \Sigma_j$, corresponding to bipartition B and assigned alleles a_1 and a_2, can be computed as

$$P((B, a_1, a_2)|\mathbf{M}) = \frac{\alpha_j(B, a_1, a_2) \cdot \beta_j(B, a_1, a_2)}{\sum_{B' \in \mathcal{B}(A_j)} \sum_{a'_1, a'_2 \in \Sigma_j} \alpha_j(B', a'_1, a'_2) \cdot \beta_j(B', a'_1, a'_2)} \tag{2.1}$$

where $\alpha_j(B, a_1, a_2)$ and $\beta_j(B, a_1, a_2)$ denote forward and backward probabilities of the state (B, a_1, a_2) and $\mathcal{B}(A_j)$, the set of all bipartitions of A_j. The above term represents the probability for a bipartition $B = (B^1, B^2)$ of the reads in A_j and alleles a_1 and a_2 assigned to these partitions. In order to finally compute the likelihood for a certain

genotype, one can marginalize over all bipartitions of a column, and all allele assignments corresponding to that genotype.

Example 2.0.1. *In order to compute genotype likelihoods for each column of the alignment matrix, posterior state probabilities corresponding to states of the same color in Figure 2.11 need to be summed up. For the first column, adding up the red probabilities gives the genotype likelihood of genotype T/T, blue of genotype G/T and yellow of G/G.*

Implementations

We created two independent software implementations of this model, one based upon WhatsHap and one from scratch, which we call MarginPhase. Each uses different optimizations and heuristics that we briefly describe.

WhatsHap implementation

We extended the implementation of WhatsHap ([35], `bitbucket.org/whatshap/whatshap`) to enable haplotype aware genotyping of bi-allelic variants based on the above model. WhatsHap focuses on re-genotyping variants, i.e. it assumes SNV positions to be given. In order to detect variants, a simple SNV calling pipeline was developed. It is based on `samtools mpileup` [47] which provides information about the bases supported by each read covering a genomic position. A set of SNV candidates is generated by selecting genomic positions at which the frequency of a non-reference allele is above a fixed threshold (0.25 for PacBio data, 0.4 for Nanopore data) and the absolute number of reads supporting the non-reference allele is at least 3.

Allele Detection. In order to construct the alignment matrix, a crucial step is to determine whether each read supports the reference or the alternative allele at each of n given genomic positions. In WhatsHap, this is done based on re-aligning sections of the reads [22]. Given an existing read alignment from the provided BAM file, its sequence in a window around the variant is extracted. It is aligned to the corresponding region of the reference sequence and additionally, to the alternative sequence, which is artificially produced by inserting the alternative allele into the reference. The alignment cost is computed by using affine gap costs. Phred scores representing the probabilities for opening and extending a gap and for a mismatch in the alignment can be estimated from the given BAM file. The allele leading to a lower alignment cost is assumed to be supported by the read and is reported in the alignment matrix. If both alleles lead to the same cost, the corresponding matrix entry is "–". The absolute difference of both alignment scores is assigned as a weight to the corresponding entry in the alignment matrix. It can be interpreted as a phred scaled probability for the allele being wrong and is utilized for the computation of output probabilities.

Read Selection. Our algorithm enumerates all bipartitions of reads covering a variant position and thus has a runtime exponential in the maximum coverage of the data. To ensure that this quantity is bounded, the same read selection step implemented previously in the WhatsHap software is run before constructing the HMM and computing genotype likelihoods. Briefly, a heuristic approach described in [48] is applied, which selects phase informative reads iteratively taking into account the number of heterozygous

variants covered by the read and its quality.

Transitions. Defining separate states for each allele assignment in $\mathbf{B_j}$ enables easy incorporation of prior genotype likelihoods by weighting transitions between states in $\mathbf{C_{j-1}}$ and $\mathbf{B_j} \times \Sigma_j \times \Sigma_j$. Since there are two states corresponding to a heterozygous genotype in the bi-allelic case (0|1 and 1|0), the prior probability for the heterozygous genotype is equally spread between these states.

In order to compute such genotype priors, the same likelihood function underlying the approaches described in [49] and [50] was utilized. For each SNV position, the model computes a likelihood for each SNV to be absent, heterozygous, or homozygous based on all reads that cover a particular site. Each read contributes a probability term to the likelihood function, which is computed based on whether it supports the reference or the alternative allele [49]. Furthermore, the approach accounts for statistical uncertainties arising from read mapping and has a runtime linear in the number of variants to be genotyped [50]. Prior genotype likelihoods are computed before read selection. In this way, information of all input reads covering a position can be incorporated.

MarginPhase Implementation

MarginPhase (`github.com/benedictpaten/marginPhase`) is an experimental, open source implementation of the described HMM written in C. It differs from the WhatsHap implementation in the method it uses to explore bipartitions and the method to generate allele support probabilities from the reads.

Read Bipartitions. The described HMM scales exponentially in terms of increasing read coverage. For typical 20-60x sequencing coverage (i.e. average number of active rows per column) it is impractical to store all possible bipartitions of the rows of the matrix. MarginPhase implements a simple, greedy pruning and merging heuristic outlined in recursive pseudocode as follows:

---

procedure COMPUTEPRUNEDHMM($\mathbf{M}$)
  if maxCov $\geq t$ **then**
    Divide $\mathbf{M}$ in half to create two matrices, $\mathbf{M_1}$ and $\mathbf{M_2}$, such that $\mathbf{M_1}$ is the first $\frac{n}{2}$ rows of $\mathbf{M}$ and $\mathbf{M_2}$ is the remaining rows of $\mathbf{M}$.
    $\mathbf{HMM_1} \leftarrow$ computePrunedHMM($\mathbf{M_1}$)
    $\mathbf{HMM_2} \leftarrow$ computePrunedHMM($\mathbf{M_2}$)
    $\mathbf{HMM} \leftarrow$ mergeHMMs($\mathbf{HMM_1}, \mathbf{HMM_2}$)
  else
    Let **HMM** be the read partitioning HMM for $\mathbf{M}$.
  return subgraph of **HMM** including visited states and transitions each with posterior probability of being visited $\geq v$, and which are on a path from the start to end nodes.

The procedure computePrunedHMM takes an alignment matrix and returns a connected subgraph of the HMM for $\mathbf{M}$ that can be used for inference, choosing to divide the input alignment matrix into two if the number of rows exceeds a threshold t, recursively.

The sub-procedure mergeHMMs takes two pruned HMMs for two disjoint alignment matrices with the same number of columns and joins them together in the natural

way such that if at each site i there are $|\mathbf{B_i^1}|$ states in $\mathbf{HMM_1}$ and $|\mathbf{B_i^2}|$ in $\mathbf{HMM_2}$ then the resulting HMM will have $|\mathbf{B_i^1}| \times |\mathbf{B_i^2}|$ states. This is illustrated in Figure 2.12. In the experiments used here $t = 8$ and $v = 0.01$.

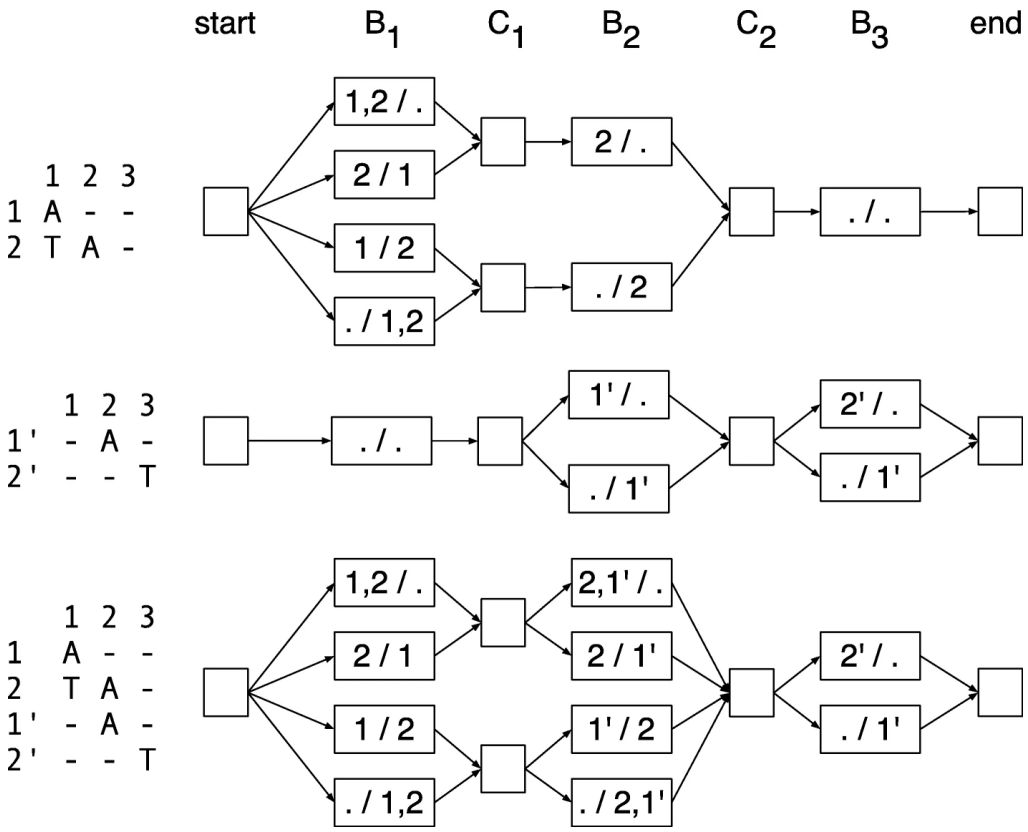

Figure 2.12: **The merger of two read partitioning HMMs with the same number of columns.** Top and middle: two HMMs to be merged; bottom: the merged HMM. Transition and emission probabilities not shown

Allele supports

In MarginPhase, the alignment matrix initially has a site for each base in the reference genome. To generate the allele support for each reference base from the reads for each read, we calculate the posterior probability of each allele (reference base) using the implementation of the banded forward-backward pairwise alignment described in [46]. The result is that for each reference base, for each read that overlaps (according to an initial guide alignment extracted from the SAM/BAM file) the reference base, we calculate the probability of each possible nucleotide (i.e., { 'A', 'C', 'G', 'T' }). The gaps are ignored and treated as missing data. This approach allows summation over all alignments within the band. Given the supports for each reference base, we then prune the set of reference bases considered to those with greater than (by default) three expected non-reference alleles. This expectation is merely the sum of non-reference allele base probabilities over the reads. This reduces the number of considered sites by approximately two orders of magnitude, greatly accelerating the HMM computation.

Substitution probabilities

We set the read error substitution probabilities, i.e., $P(M_{i,j}|(H_j)^k$ empirically and iteratively. Starting from a 1% flat substitution probability, we generate a ML read bipartition and pair of haplotypes, we then re-estimate the read error probabilities from the differences between the reads and the haplotypes. We then rerun the model and repeat the process to derive the final probabilities. For the haplotype substitution probabilities, i.e., $P((H_j)^k|Z_j)$, we use substitution probabilities of 0.1% for transver-

sions and 0.4% for transitions, reflecting the facts that transitions are twice as likely empirically but that there are twice as many possible transversions.

Phase blocks

MarginPhase divides the read partitioning HMMs into phase sets based on the number of reads which span adjacent likely heterozygous sites. The bipartitioning is performed on each of these phase sets individually. MarginPhase's output includes a BAM which encodes the phasing of each read, including which phase set it is in, which haplotype it belongs to, and what of the aligned portion falls into each phase set. Reads which span a phase set boundary have information for both encoded in them.

Chapter 3

Genome Assembly

Background

Reference genomes and their biases

Many genomics analyses, like variant calling, are performed relative to a reference genome for the species, such as GRCh38 for humans. Using a single reference genome has many benefits, such as providing consistent genomic coordinates used in all annotations and analyses. However, there are some situations that require the assembly of a new genome. If a species of interest does not already have a reference genome, or there is a desire to improve upon the existing reference due to technological advancements, or there is some other reason for not using an existing reference like a desire to prevent reference bias in downstream applications, the genome will need to be assembled from scratch. *De novo* genome assembly is the process of assembling a genome without any guidance from an existing reference genome. In addition to increasing the

number of species with assembled genomes, *de novo* genome assembly can be useful for creating reference assemblies for specific subpopulations, and characterizing structural variations that can be difficult to identify when comparing to a reference that is vastly different in the region of interest.

Genome assembly techniques

Genome assembly is the process of reconstructing an individual's genome, given DNA sequencing reads from that individual. Due to the limitations of sequencing technologies, sequenced reads are much shorter than the longest stretches of contiguous DNA in a genome. The reads commonly used in the field today can range in size anywhere from 150 base pairs per read for Illumina shotgun sequencing, to over 1 million bases with Oxford nanopore reads, but even these longest reads are still a fraction of the size of many chromosomes. Humans, for example, have some chromosomes that are over 200 million base pairs long. Therefore, *de novo* genome assembly requires using the reads and their overlapping subsequences to reconstruct the original sequences of the chromosomes from which they originated. The algorithms for genome assembly differ greatly depending on the type of sequencing reads used and their different error distributions and read lengths.

The first approaches to genome assembly used Sanger sequencing on genomic fragments cloned into bacterial artificial chromosomes (BACs), including the first assembly of the human genome in the Human Genome Project [51]. These assemblies from Sanger sequencing were accurate on a base-pair level, but incredibly expensive to

create. They also contained many gaps of unassembled sequence between contigs, which are continuous blocks of assembled DNA. After the introduction of considerably less expensive next-generation short read technologies, it was possible to create many more genome assemblies, though these assemblies were not very contiguous. Contig N50s, or the length of the shortest contig that covers 50 percent of the total length of the genome when contigs are ordered in decreasing size, are very low in short read assemblies, in the 20-100 kilobase range [52, 53]. For example, the initial mouse assembly created with Illumina reads had a contig N50 of 24.8 kb [54]. The accuracy of the short assembled portions is fairly high, at least.

Newer DNA sequencing technologies, such as long reads and linked reads, can be used to create highly contiguous genome assemblies. Longer reads allow for finding longer overlaps between reads, and make it easier to unambiguously extend contigs. This results in much higher contig N50s, in the range of tens of megabases rather than tens of kilobases [23, 55, 56]. However, the quality scores of the assembled contigs are lower, particularly before any additional polishing of the assembly. Long read assemblers will be discussed in greater detail later, particularly our own assembler for nanopore reads, Shasta [23].

Assembly algorithms

Current assemblers tend to fall into two major categories. The first type are Overlap-Layout-Consensus (OLC) assemblers, which as can be inferred by the name, assemble genomes using three main steps: finding overlaps between the reads and build-

ing an overlap graph, using the overlaps to group sections of the graph into contigs, and building a consensus sequence of the most likely nucleotide sequence for each contig. In the overlap graph, reads are represented as nodes in the graph, and the overlaps between reads as edges. OLC assemblers can be used on many types of sequencing reads, both short and long. However, OLC assemblers can be computationally inefficient due to the overlap stage, where many alignments between pairs of reads need to be made to compute the overlaps and construct the graph.

The second type are De Bruijn Graph (DBG) assemblers. This type of assembler revolves around constructing a de Bruijn graph out of the input reads. The reads are decomposed into their constituent kmers. For example, the 4-mers that make up the sequence "GATTACA" would be GATT, ATTA, TTAC, and TACA. The de Bruijn graph is built using kmers as nodes, with edges between overlapping kmers, and reads are paths through the graph. DBG assemblers require highly accurate sequencing reads. However, if a de Bruijn graph can be used, this type of assembler can be very computationally efficient, as constructing a de Bruijn graph is much faster than computing the overlaps between pairs of reads within the input dataset.

Finishing steps for an assembly

Scaffolding

After producing an initial contig assembly, the contigs can be linked together further to create larger scaffolds that organize contigs into larger structures that are hopefully closer to the size of chromosomes. Proximity ligation reads are particularly

useful for this, as each pair of reads represents sequences that are in close physical proximity to each other within the nucleus, so contigs that have large amount of contacts between them can be scaffolded together. Some available scaffolding applications that use proximity ligation reads such as HiC and Chicago are HiRise [57] and SALSA2 [58]. Gaps created by scaffolding can be filled with long reads, using a tool like PBJelly2 [59] which uses PacBio reads.

Polishing

Finally, assemblies can be polished by aligning sequencing reads back to the assembly, and fixing errors where the read alignments disagree with the assembled sequencing. Accurate short reads have been traditionally used for polishing, using a program like Pilon [60]. Alternatively, the long reads themselves can be used to polish assemblies. Later in this chapter, we introduce our own polishing software tools developed in the lab (MarginPolish and HELEN) that use nanopore reads to polish.

Use of multiple sequencing types for assembly

Integrating multiple different DNA sequencing types during an assembly workflow can improve assemblies by taking advantage of the different characteristics of each sequencing type. An example of an assembly workflow involving many options for sequencing types is shown in Figure 3.1.

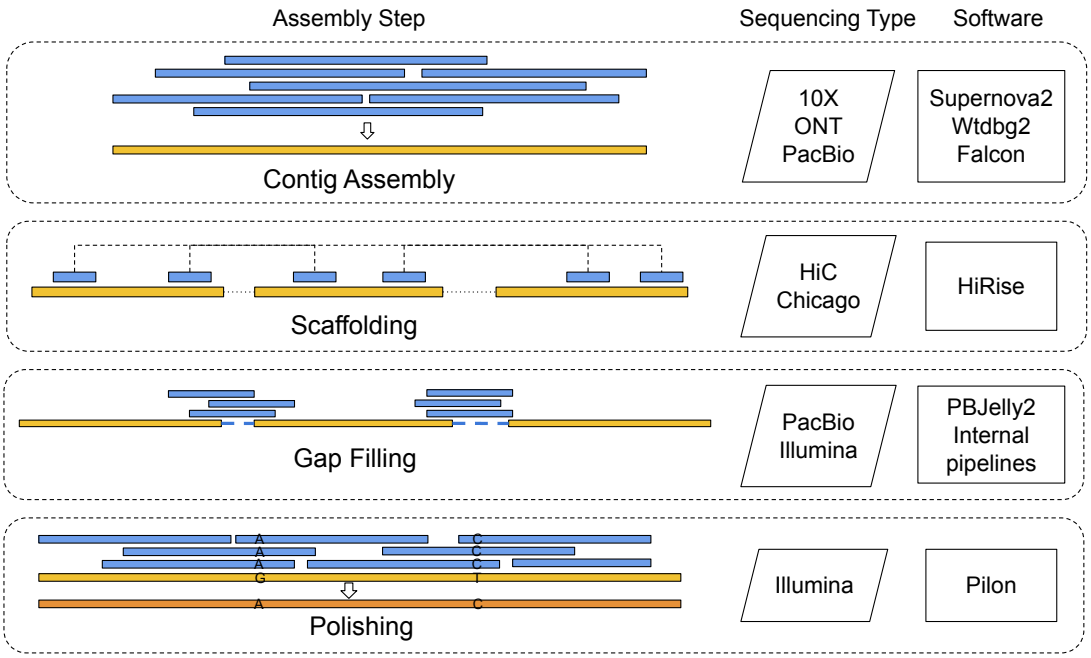

Figure 3.1: **Assembly pipeline.** An overview of an example haploid consensus assembly pipeline which uses multiple sequencing technologies and pieces of software. This is non-exhaustive, as additional sequencing types and software could be included.

Diploid genome assembly

Ignored up until this point is another major limitation of the current state of assembly: all of these assemblers produce a single, collapsed haplotype. This results in the loss of information for a diploid (or higher ploidy) organism, and the more heterozygous the individual is, the greater the loss of information will be. There are some assemblers previously mentioned that attempt to address this by creating varying levels of "pseudohaploid" assemblies, where alternative haplotype paths can be represented for portions of the genome, such as Supernova2 which works with 10X linked reads.

However, these phased segments do not cover the whole genome, and there is no notion of the long-range phasing of these haplotypes.

ONT reads and assembly

Nanopore sequencing, commercialized by Oxford Nanopore Technologies (ONT), is particularly apt for use in genome assembly. Nanopore reads can be very long, over 100 kilobases in length when using the ultra-long sequencing protocol [33], which facilitates the assembly of the most challenging regions of the genome, such as centromeres, acrocentric short arms, telomeres, rRNAs, and segmental duplications. Nanopore sequencing has the downside of being less accurate than other long read technology (PacBio). Its errors are concentrated in homopolymers, leading to a greater rate of homopolymer errors in resulting assemblies. However, it does have benefits because in addition to having the longest read lengths of any sequencing technology, it can also be cheaper and faster to generate the large quantities of input data needed for genome assembly using high throughput sequencers like the PromethION.

There are existing methods for assembling genomes *de novo* using nanopore reads, but they either require a lot of computing resources, or are not very accurate. For example, a *de novo* assembly of NA12878 with nanopore reads used Canu [55] for the assembly, and took over 150,000 CPU hours and weeks of wall-clock time [33]. Canu, which is fairly accurate, is partially so slow because it performs an error correction step, where reads are corrected in order to minimize the base-level errors present. Other methods which are faster than Canu, such as wtdbg2 [61] and Flye [62], are still relatively

slow (taking days of wall clock time as opposed to weeks), and produce less accurate assemblies in part because they do not include an error correction step. Improvements to the speed and accuracy of software for assembling and polishing assemblies using nanopore reads would be beneficial for the field and could enable the sequencing of many large genomes quickly, at low cost.

Shasta assembler

The Shasta assembler, developed primarily by Paolo Carnevali while he worked for the Chan Zuckerberg Initiative, with contributions by other lab members such as Ryan Lorig-Roach, aims to reduce the amount of time needed to assemble genomes using nanopore reads by orders of magnitude, while obtaining assemblies of better quality (or at least comparable to) existing assemblers.

A few key points of Shasta's methods are the following. Shasta uses run-length encoding in order to minimize the effect of errors in homopolymers typically observed in the error profiles of nanopore reads. To increase the speed of assembly, Shasta uses a marker representation of reads, where each read is represented as a sequence of fixed markers observed in the read, and markers are determined by a fraction of all kmers for a given value k. Shasta runs a modified MinHash algorithm to find candidate pairs of overlapping reads, then computes alignments between candidates and creates a Marker Graph. Data structures are all stored in memory, to reduce time spent reading and writing from disk, and consequently, Shasta needs to be run on a single node with a

large amount of memory (1-2 TB for a mammal sized assembly).

Future directions: PacBio HiFi reads and assembly

After the publication of Shasta, PacBio came out with a new type of high-fidelity reads, HiFi for short, that are produced using their Circular Consensus Sequencing (CCS) mode. These reads are much more accurate than the traditional PacBio CLR read. Because the accuracy of the PacBio CLR reads was a serious limitation to using them for genome assembly, HiFi reads are much better suited to the problem. Hifiasm [63] is an example of an assembler that works on HiFi reads and produces assemblies that are both highly contiguous and have high quality scores. HiFi reads are still limited by their relatively smaller read lengths compared to nanopore reads. However, combining HiFi and nanopore reads together by creating an initial contig assembly with the HiFi reads and threading the nanopore reads through after to use the long range information is a promising strategy. This strategy is employed in an automated manner by Verkko (not yet published).

Contributions

The following sections include most of the full text of the paper published for Shasta, "Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes" published in Nature Biotechnology in May 2020 [23]. I was a co-first author of this paper, along with Kishwar Shafin, Trevor Pesout, Ryan

Lorig-Roach, and Hugh Olsen. The goal of this paper was to demonstrate the use of the Shasta assembly software to sequence and assemble eleven human genomes in approximately eleven days. This paper contains four different projects: the generation of the ONT sequencing reads using a new ultralong read length protocol that aimed to increase the number of read lengths greater than 100kb, the assembler (Shasta) used to create the initial contig assemblies, the polisher (MarginPolish) which was built upon the previously introduced 'Margin' framework, and the polisher (HELEN) that uses a recurrent neural network. We also performed proximity ligation sequencing with HiC reads, and I used those to scaffold the assemblies. We compared Shasta to three other existing assemblers, and the polishers to two existing polishers.

I helped design and run the assembly pipelines for Shasta (and the other assemblers), as well as the polishing and scaffolding pipelines. I compared Shasta against the other assemblers (and the polishers against other polishers) and performed many of the quality analysis portions of the pipeline. I also designed and made several figures and tables and wrote large portions of the main text.

I moved several portions of the main text to the appendix of this dissertation, as they did not directly relate to the work I personally did on this manuscript, but are still important for understanding the paper. These sections include some of the methods for the new software tools presented in the paper that I did not help develop.

Future directions post publication

Shasta continues to be in development, with a goal of increasing the contiguity of assemblies to reach telomere-to-centromere contiguity (and eventually, telomere-to-telomere). Diploid genome assembly is another goal for Shasta. However, it is no longer funded by CZI. Since the initial publication of the results, N50s of the assemblies have increased by a factor of up to 2-4 times, largely due to the work of detangling repetitive parts of the genome such as segmental duplications.

Shasta was also tested in the Human Pangenome Reference Consortium (HPRC), whose goal is to eventually create 350 reference-quality assemblies of a genetically diverse group of people. This would test if Shasta is truly scalable and able to compete with assemblers using more accurate sequencing types, such as PacBio HiFi. Unfortunately, the assemblers that used HiFi reads were found to be superior, as the base-level quality is simply so much higher.

Full text of paper

Abstract

De novo assembly of a human genome using nanopore long-read sequences has been reported, but it used more than 150,000 CPU hours and weeks of wall-clock time. To enable rapid human genome assembly, we present Shasta, a de novo long-read assembler, and polishing algorithms named MarginPolish and HELEN. Using a single

PromethION nanopore sequencer and our toolkit, we assembled 11 highly contiguous human genomes de novo in 9 days. We achieved roughly 63x coverage, 42 kb read N50 values and 6.5x coverage in reads > 100 kb using three flow cells per sample. Shasta produced a complete haploid human genome assembly in under 6 hours on a single commercial compute node. MarginPolish and HELEN polished haploid assemblies to more than 99.9% identity (Phred quality score QV = 30) with nanopore reads alone. Addition of proximity-ligation sequencing enabled near chromosome-level scaffolds for all 11 genomes. We compare our assembly performance to existing methods for diploid, haploid and trio-binned human samples and report superior accuracy and speed.

Introduction

Reference-based methods such as GATK[13] can infer human variations from short-read sequences, but the results only cover ˜90% of the reference human genome assembly [2, 3]. These methods are accurate with respect to single-nucleotide variants and short insertions and deletions (indels) in this mappable portion of the reference genome[64]. However, it is difficult to use short reads for de novo genome assembly [53], to discover structural variations (SVs)[65, 66] (including large indels and base-level resolved copy number variations), or to resolve phasing relationships without exploiting transmission information or haplotype panels8.

Third generation sequencing technologies, including linked-reads[67, 68, 69] and long-read technologies[70, 71], overcome the fundamental limitations of short-read

sequencing for genome inference. In addition to increasingly being used in reference guided methods[2, 72, 73, 39], long-read sequences can generate highly contiguous de novo genome assemblies[74].

Nanopore sequencing, as commercialized by Oxford Nanopore Technologies (ONT), is particularly useful for de novo genome assembly because it can produce high yields of very long 100+ kilobase (kb) reads[33]. Very long reads hold the promise of facilitating contiguous, unbroken assembly of the most challenging regions of the human genome, including centromeric satellites, acrocentric short arms, ribosomal DNA arrays and recent segmental duplications[75, 76, 77]. The de novo assembly of a nanopore sequencing based human genome has been reported[33]. This earlier effort needed 53 ONT MinION flow cells and the assembly required more than 150,000 CPU hours and weeks of wall-clock time, quantities that are unfeasible for high throughput human genome sequencing efforts.

To enable easy, cheap and fast de novo assembly of human genomes we developed a toolkit for nanopore data assembly and polishing that is orders of magnitude faster than state-of-the-art methods. We use a combination of nanopore and proximity-ligation (HiC) sequencing[67] and our toolkit, and we report improvements in human genome sequencing coupled with reduced time, labor and cost.

Results

Eleven human genomes sequenced in 9 days

We selected for sequencing 11, low-passage (six passages), human cell lines of the offspring of parent-child trios from the 1,000 Genomes Project[78] and genome-in-a-bottle (GIAB)[40] sample collections. Samples were selected to maximize captured allelic diversity (see Methods).

We carried out PromethION nanopore sequencing and HiC Illumina sequencing for the 11 genomes. We used three flow cells per genome, with each flow cell receiving a nuclease flush every 20–24 h. This flush removed long DNA fragments that could cause the pores to become blocked over time. Each flow cell received a fresh library of the same sample after the nuclease flush. A total of two nuclease flushes were performed per flow cell, and each flow cell received a total of three sequencing libraries. We used Guppy v.2.3.5 with the high accuracy flipflop model for basecalling (see Methods).

Nanopore sequencing of all 11 genomes took 9 days and produced 2.3 terabases (Tb) of sequence. We ran up to 15 flow cells in parallel during these sequencing runs. Results are shown in Fig. 3.2 and Supplementary Tables 1-3. Nanopore sequencing yielded an average of 69 gigabases (Gb) per flow cell, with the total throughput per individual genome ranging between 48x (158 Gb) and 85x (280 Gb) coverage per genome (Fig. 3.2a). The read N50 values for the sequencing runs ranged between 28 and 51 kb (Fig. 3.2b). (An N50 value is a weighted median; it is the length of the sequence in a set for which all sequences of that length or greater sum to 50% of the set's total size.)

We aligned nanopore reads to the human reference genome (GRCh38) and calculated their alignment identity to assess sequence quality (see Methods). We observed that the median and modal alignment identity was 90 and 93%, respectively (Fig. 3.2c). The sequencing data per individual genome included an average of 55x coverage arising from $>$ 10-kb reads and 6.5x coverage from $>$ 100-kb reads (Fig. 3.2d). This was in large part due to size selection that yielded an enrichment of reads longer than 10 kb. To test the generality of our sequencing methodology for other samples, we sequenced high-molecular weight DNA isolated from a human saliva sample using identical sample preparation. The library was run on a MinION (roughly one-sixth the throughput of a ProMethION flow cell) and yielded 11 Gb of data at a read N50 of 28 kb (Supplementary Table 4), extrapolating both are within the lower range achieved with cell-line derived DNA.

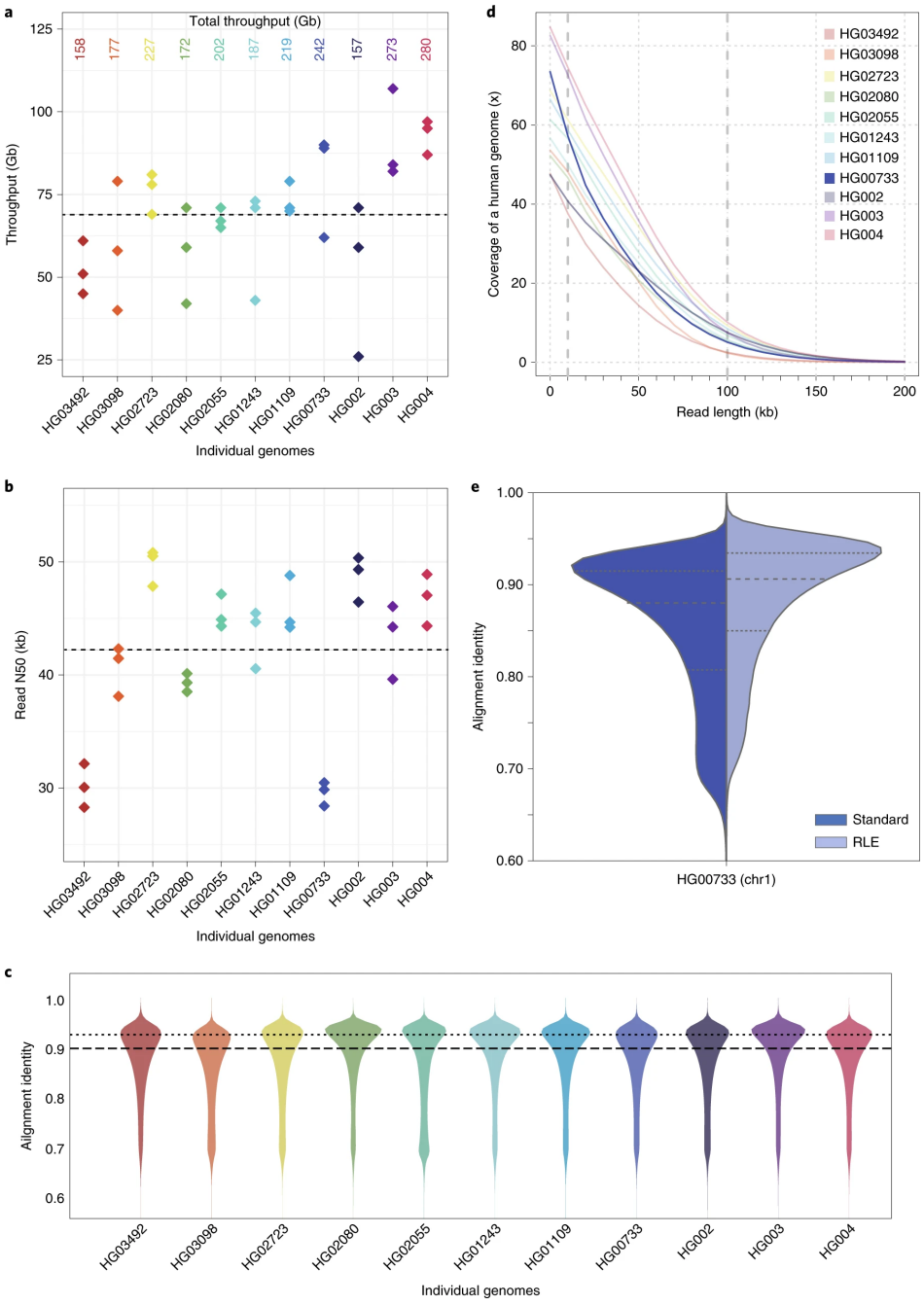

Figure 3.2: **Nanopore sequencing data.** a, Throughput in gigabases from each of three flow cells for 11 samples, with total throughput at top. Each point is a flow cell. b, Read N50 values for each flow cell. Each point is a flow cell. c, Alignment identities against GRCh38. Medians in a–c shown by dashed lines, dotted line in c is the mode. Each line is a single sample comprising three flow cells. d, Genome coverage as a function of read length. Dashed lines indicate coverage at 10 and 100 kb. HG00733 is accentuated in dark blue as an example. Each line is a single sample comprising three flow cells. e, Alignment identity for standard and RLE reads. Data for HG00733 chromosome 1 flow cell 1 are shown (4.6 Gb raw sequence). Dashed lines denote quartiles.

Shasta assembler for long sequence reads

Shasta was designed to be orders of magnitude faster and cheaper at assembling a human-scale genome from nanopore reads than the Canu assembler used in our earlier work [33]. During most Shasta assembly phases, reads are stored in a homopolymer-compressed form using run-length encoding (RLE)[79, 80, 81]. In this form, identical consecutive bases are collapsed, and the base and repeat count are stored. For example, `GATTTACCA` would be represented as (`GATACA`, `113121`). This representation is insensitive to errors in the length of homopolymer runs, thereby addressing the dominant error mode for Oxford Nanopore reads[70]. As a result, assembly noise due to read errors is decreased, and notably higher identity alignments are facilitated (Fig. 3.2e). A marker representation of reads is also used, in which each read is represented as the sequence of occurrences of a predetermined, fixed subset of short k-mers (marker representation) in its run-length representation. A modified *MinHash* [82, 83] scheme is used to find candidate pairs of overlapping reads, using as *MinHash* features consecutive occurrences of m markers (default m = 4). Optimal alignments in marker representation are computed for all candidate pairs. The computation of alignments in marker representation is very efficient, particularly as various banded heuristics are used. A *marker graph* is created in which each vertex represents a marker found to be aligned in a set of several reads. The marker graph is used to assemble sequence after undergoing a series of simplification steps. The assembler runs on a single machine with a large amount of memory (typically 1–2 Tb for a human assembly). All data structures are kept in

memory, and no disk I/O takes place except for initial loading of the reads and final output of assembly results.

Benchmarking Shasta

We compared Shasta to three contemporary assemblers: Wtdbg2[61], Flye[62], and Canu[55]. We ran all four assemblers on available read data from two diploid human samples, HG00733 and HG002, and one haploid human sample, CHM13. HG00733 and HG002 were part of our collection of 11 samples, and data for CHM13 came from the T2T consortium[84].

Canu consistently produced the most contiguous assemblies, with contig NG50 values of 40.6, 32.3 and 79.5 Mb, for samples HG00733, HG002 and CHM13, respectively (Fig. 3.3a). (NG50 is similar to N50, but for 50% of the estimated genome size.) Flye was the second most contiguous, with contig NG50 values of 25.2, 25.9 and 35.3 Mb, for the same samples. Shasta was next with contig NG50 values of 21.1, 20.2 and 41.1 Mb. Wtdbg2 produced the least contiguous assemblies, with contig NG50 values of 15.3, 13.7 and 14.0 Mb.

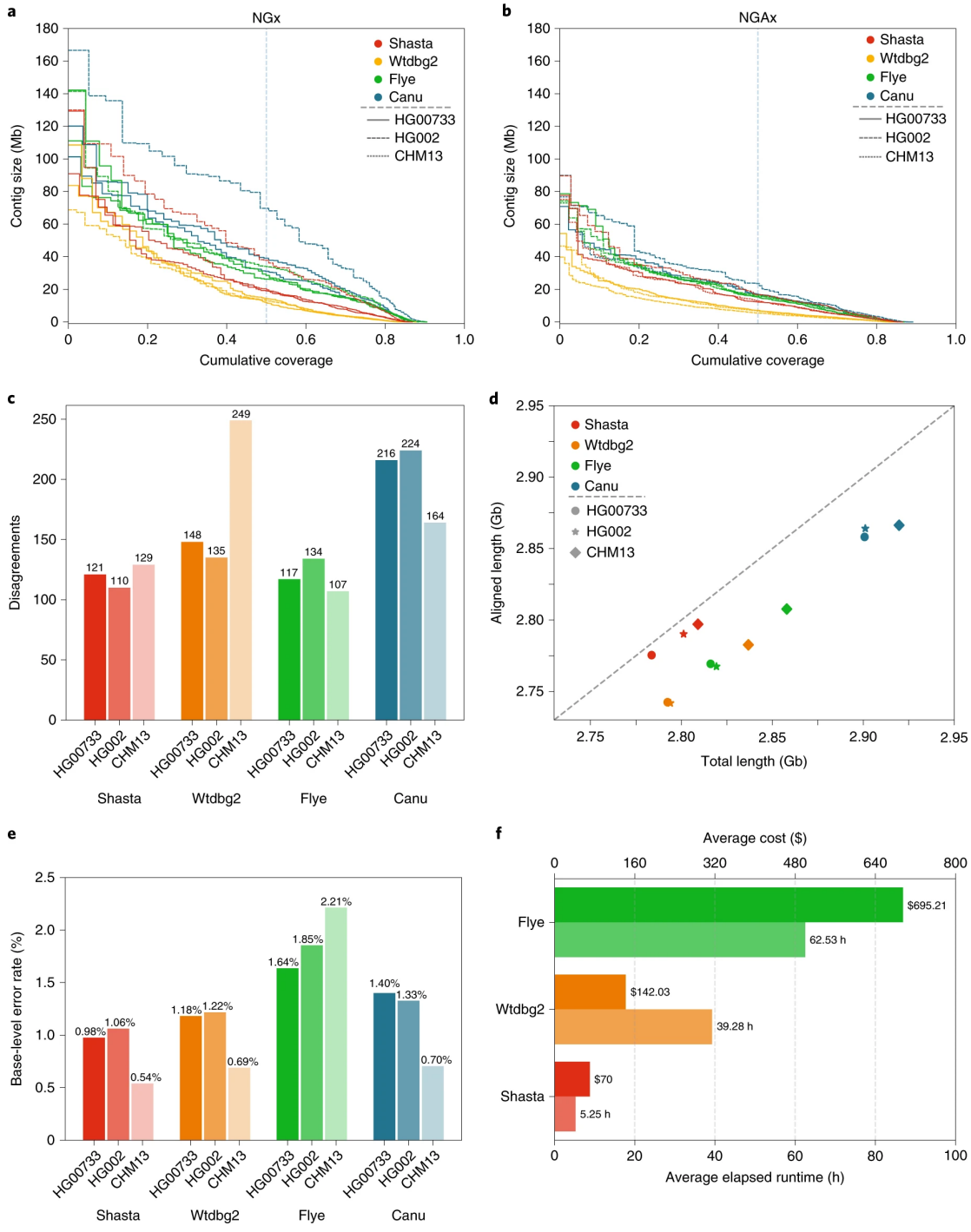

Figure 3.3: **Assembly metrics for Shasta, Wtgdb2, Flye and Canu before polishing.** a, NGx plot showing contig length distribution. The intersection of each line with the dashed line is the NG50 for that assembly. b, NGAx plot showing the distribution of aligned contig lengths. Each horizontal line represents an aligned segment of the assembly unbroken by a disagreement or unmappable sequence with respect to GRCh38. The intersection of each line with the dashed line is the aligned NGA50 for that assembly. c, Assembly disagreement counts for regions outside centromeres, segmental duplications and, for HG002, known SVs. d, Total generated sequence length versus total aligned sequence length (against GRCh38). e, Balanced base-level error rates for assembled sequences. f, Average runtime and cost for assemblers (Canu not shown).

Conversely, aligning the samples to GRCh38 and evaluating with QUAST[85], Shasta had between 4.2 and 6.5x fewer disagreements (locations where the assembly contains a breakpoint with respect to the reference assembly) per assembly than the other assemblers (Supplementary Table 5). Breaking the assemblies at these disagreements and unaligned regions with respect to GRCh38, we observe much smaller absolute variation in contiguity (Fig. 3.3b and Supplementary Table 5). However, a substantial fraction of the identified disagreements likely reflect true SVs with respect to GRCh38. To address this, we discounted disagreements within chromosome Y, centromeres, acrocentric chromosome arms, QH-regions and known recent segmental duplications (all of which are enriched in SVs[86, 87]); in the case of HG002, we further excluded a set of known SVs[88]. We still observe between 1.2x and 2x fewer disagreements in Shasta relative to Canu and Wtdbg2, and comparable results against Flye (Fig. 3.3c and Supplementary Table 6). To account for differences in the fraction of the genomes assembled, we analyzed disagreements contained within the intersection of all the assemblies (that is, in regions where all assemblers produced a unique assembled sequence). This produced results highly consistent with the previous analysis and suggests Shasta and Flye have the lowest and comparable rates of misassembly (Methods, see Supplementary Table 7). Finally, we used QUAST to calculate disagreements between the T2T Consortium's chromosome X assembly, a highly curated, validated assembly[84] and the subset of each CHM13 assembly mapping to it; Shasta has two to 17 times fewer disagreements than the other assemblers while assembling almost the same fraction of the assembly (Supplementary Table 8).

Canu consistently assembled the largest genomes (average 2.91 Gb), followed by Flye (average 2.83 Gb), Wtdbg2 (average 2.81 Gb) and Shasta (average 2.80 Gb). Due to their similarity, we would expect the most of these assembled sequences to map to another human genome. Discounting unmapped sequence, the differences are smaller: Canu produced an average of 2.86 Gb of mapped sequence per assembly, followed by Shasta (average 2.79 Gb), Flye (average 2.78 Gb) and Wtdbg2 (average 2.76 Gb) (Fig. 3.3d, see Methods). This analysis supports the notion that Shasta is currently relatively conservative versus its peers, producing the highest ratio of directly mapped assembly per sample.

For HG00733 and CHM13 we examined a library of bacterial artificial chromosome (BAC) assemblies (Methods). The BACs were largely targeted at known segmental duplications (473 of 520 BACs lie within 10 kb of a known duplication). Examining the subset of BACs for CHM13 and HG00733 that map to unique regions of GRCh38 (see Methods), we find Shasta contiguously assembles all 47 BACs, with Flye performing similarly (Supplementary Table 9). In the full set, we observe that Canu (411) and Flye (282) contiguously assemble a larger subset of the BACs than Shasta (132) and Wtdbg2 (108), confirming the notion that Shasta is relatively conservative in these duplicated regions (Supplementary Table 10). Examining the fraction of contiguously assembled BACs of all BACs represented in each assembly we can measure an aspect of assembly correctness. In this regard Shasta (97%) produces a much higher percentage of correct BACs in duplicated regions versus its peers (Canu 92%, Flye 87%, Wtdbg2 88%). In the intersected set of BACs attempted by all assemblers (Supplementary Table 11), Shasta,

100%; Flye, 100%; Canu, 98.50% and Wtdbg2, 90.80% all produce comparable results.

Shasta produced the most base-level accurate assemblies (average balanced error rate 0.98% on diploid and 0.54% on haploid), followed by Wtbdg2 (1.18% on diploid and 0.69% on haploid), Canu (1.40% on diploid and 0.71% on haploid) and Flye (1.64% on diploid and 2.21% on haploid) (Fig. 3.3e, see Methods and Supplementary Table 12). We also calculated the base-level accuracy in regions covered by all the assemblies and observe results consistent with the whole-genome assessment (Supplementary Table 13).

Shasta, Wtdbg2 and Flye were run on a commercial cloud, allowing us to reasonably compare their cost and runtime (Fig. 3.3e, see Methods). Shasta took an average of 5.25 h to complete each assembly at an average cost of US$70 per sample. In contrast, Wtdbg2 took 7.5× longer and cost 3.7× as much, and Flye took 11.9× longer and cost 9.9× as much. Due to the anticipated cost and complexity of porting it to Amazon Web Services (AWS), the Canu assemblies were run on a large, institutional compute cluster, consuming up to US$19,000 (estimated) of compute and took around 4–5 d per assembly (Methods, see Supplementary Tables 14 and 15).

To assess the use of using Shasta for SV characterization we created a workflow to extract putative heterozygous SVs from Shasta assembly graphs (Methods). Extracting SVs from an assembly graph for HG002, the length distribution of indels shows the characteristic spikes for known retrotransposon lengths (Supplementary Fig. 1). Comparing these SVs to the high-confidence GIAB SV set we find good concordance, with a combined F1 score of 0.68 (Supplementary Table 16).

Contiguously assembling major histocompatibility complex (MHC) haplotypes

The MHC region is difficult to resolve using short reads due to its repetitive and highly polymorphic nature [89], and recent efforts to apply long-read sequencing to this problem have shown promise[33, 90]. We analyzed the assemblies of CHM13 and HG00733 to see if they spanned the MHC region. For the haploid assemblies of CHM13 we find MHC is entirely spanned by a single contig in all four assemblers' output, and most closely resembles the GL000251.2 haplogroup among those provided in GRCh38 (Fig. 3.4a, Supplementary Fig. 2 and Supplementary Table 17). In the diploid assembly of HG00733 two contigs span most of the MHC for Shasta and Flye, while Canu and Wtdbg2 span the region with one contig (Fig. 3.4b and Supplementary Fig. 3). However, we note that all these chimeric diploid assemblies lead to sequences that do not closely resemble any haplogroup (Methods).

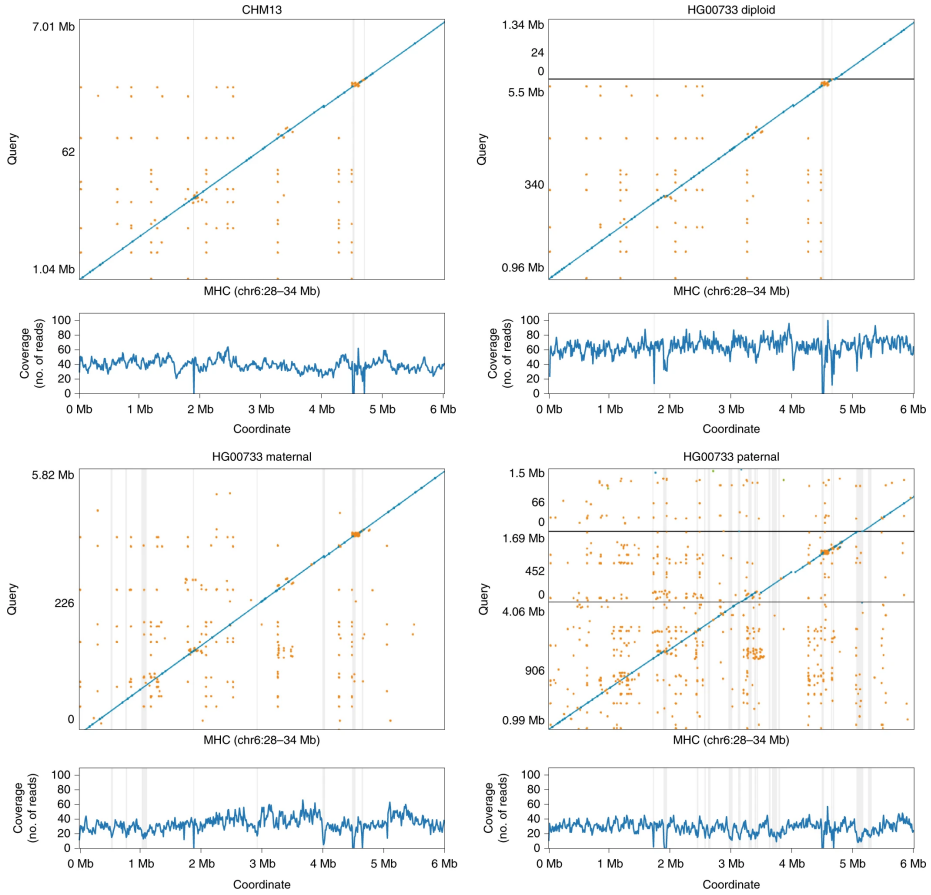

Figure 3.4: **Shasta MHC assemblies compared with the reference human genome.** Unpolished Shasta assembly for CHM13 and HG00733, including HG00733 trio-binned maternal and paternal assemblies. Shaded gray areas are regions in which coverage (as aligned to GRCh38) drops below 20. Horizontal black lines indicate contig breaks. Blue and green describe unique alignments (aligning forward and reverse, respectively) and orange describes multiple alignments.

To attempt to resolve haplotypes of HG00733 we used trio-binning[91] to partition the reads for HG00733 into two sets based on likely maternal or paternal lineage and assembled the haplotypes (Methods). For all assemblers and each haplotype assembly, the global contiguity worsened substantially (as the available read data coverage

was approximately halved and, further, not all reads could be partitioned), but the resulting disagreement count decreased (Supplementary Table 18). When using haploid trio-binned assemblies, the MHC was spanned by a single contig for the maternal haplotype (Fig. 3.4c, Supplementary Fig. 4 and Supplementary Table 19), with high identity to GRCh38 and having the greatest contiguity and identity with the GL000255.1 haplotype. For the paternal haplotype, low coverage led to discontinuities (Fig. 3.4d) breaking the region into three contigs.

Deep neural network-based polishing for long-read assemblies

We developed a deep neural network-based consensus sequence polishing pipeline designed to improve the base-level quality of the initial assembly. The pipeline consists of two modules: MarginPolish and the homopolymer encoded long-read error-corrector for Nanopore (HELEN). MarginPolish uses a banded form of the forward–backward algorithm on a pairwise hidden Markov model (pair-HMM) to generate pairwise alignment statistics from the RLE alignment of each read to the assembly. From these statistics, MarginPolish generates a weighted RLE partial order alignment (POA)[92] graph that represents potential alternative local assemblies. MarginPolish iteratively refines the assembly using this RLE POA, and then outputs the final summary graph for consumption by HELEN. HELEN uses a multi-task recurrent neural network (RNN)[93] that takes the weights of the MarginPolish RLE POA graph to predict a nucleotide base and run length for each genomic position. The RNN takes advantage of contextual genomic features and associative coupling of the POA weights to the correct base and

run length to produce a consensus sequence with higher accuracy.

To demonstrate the effectiveness of MarginPolish and HELEN, we compared them with the state-of-the-art nanopore assembly polishing workflow: four iterations of Racon polishing[94] followed by Medaka[95]. MarginPolish is analogous in function to Racon, both using pair-HMM-based methods for alignment and POA graphs for initial refinement. Similarly, HELEN is analogous to Medaka, in that both use a deep neural network and both work from summary statistics of reads aligned to the assembly.

Figure 3.5a and Supplementary Tables 20–22 detail error rates for the four methods performed on the HG00733 and CHM13 Shasta assemblies (see Methods) using Pomoxis [96]. For the diploid HG00733 sample MarginPolish and HELEN achieve a balanced error rate of 0.388% (Phred quality score QV = 24.12), compared to 0.455% (QV = 23.42) by Racon and Medaka. For both polishing pipelines, a notable fraction of these errors are likely due to true heterozygous variations. For the haploid CHM13 we restrict comparison to the highly curated X chromosome sequence provided by the T2T consortium[84]. We achieve a balanced error rate of 0.064% (QV = 31.92), compared to Racon and Medaka's 0.110% (QV = 29.59).

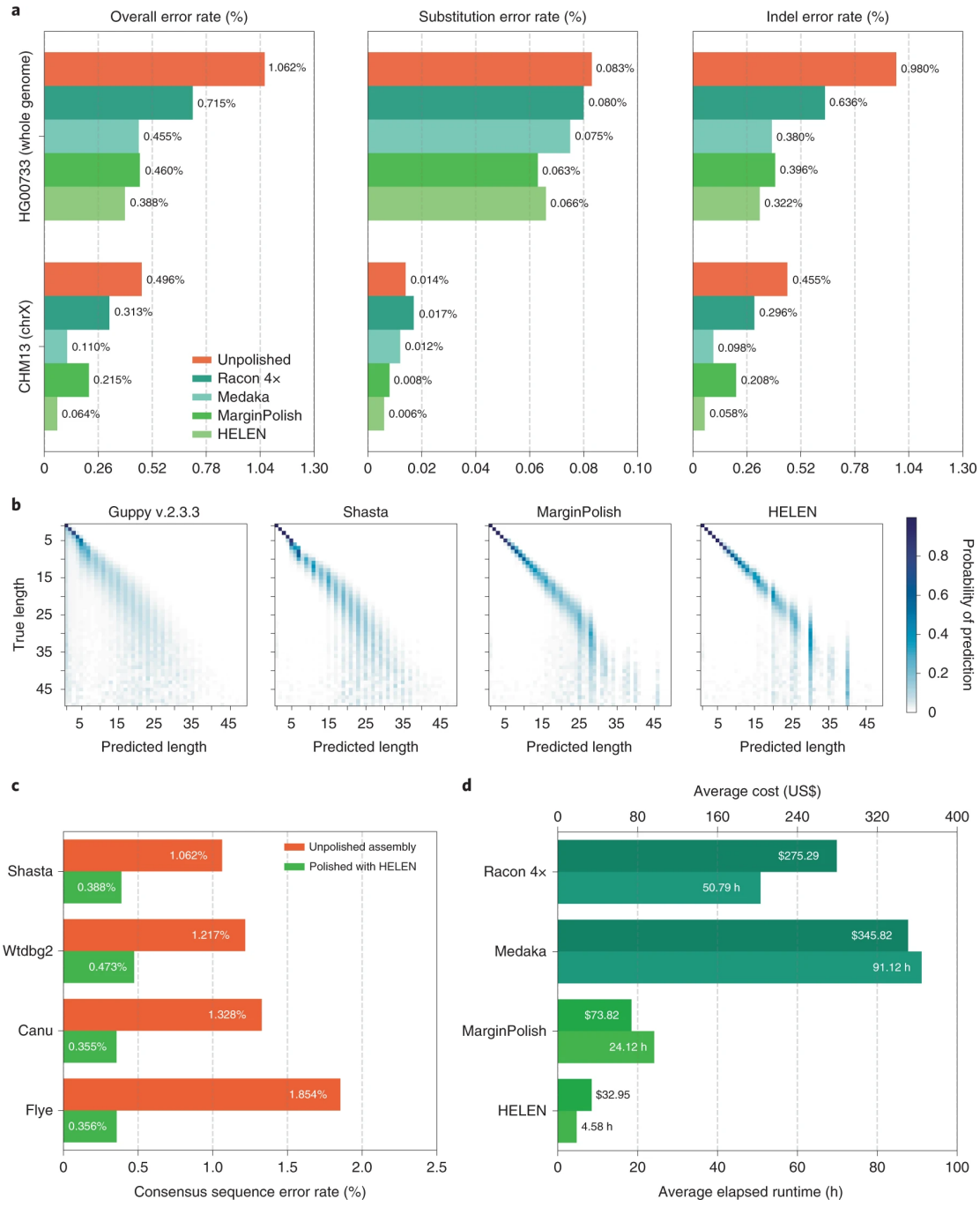

Figure 3.5: **Polishing assembled genomes.** a, Balanced error rates for the four methods on HG00733 and CHM13. b, Row-normalized heatmaps describing the predicted run lengths (x axis) given true run lengths (y axis) for four steps of the pipeline on HG00733. Guppy v.2.3.3 was generated from 3.7 Gb of RLE sequence. Shasta, MarginPolish and HELEN were generated from whole assemblies aligned to their respective truth sequences. c, Error rates for MarginPolish and HELEN on four assemblies. d, Average runtime and cost.

For all assemblies, errors were dominated by indel errors; for example, substitution errors are 3.16 and 2.9 times fewer than indels in the polished HG000733 and CHM13 assemblies, respectively. Many of these errors relate to homopolymer length confusion; Fig. 3.5b analyzes the homopolymer error rates for various steps of the polishing workflow for HG00733. Each panel shows a heatmap with the true length of the homopolymer run on the y axis and the predicted run length on the x axis, with the color describing the likelihood of predicting each run length given the true length. Note that the dispersion of the diagonal steadily decreases. The vertical streaks at high run lengths in the MarginPolish and HELEN confusion matrix are the result of infrequent numerical and encoding artifacts (see Methods and Supplementary Fig. 5).

Figure 3.5c and Supplementary Table 23 show the overall error rate after running MarginPolish and HELEN on HG00733 assemblies generated by different assembly tools, demonstrating that they can be usefully employed to polish assemblies generated by other tools.

To investigate the benefit of using short reads for further polishing, we polished chromosome X of the CHM13 Shasta assembly after MarginPolish and HELEN using 10X Chromium reads with the Pilon polisher[97]. This led to a roughly twofold reduction in base errors, increasing the QV from roughly 32 (after polishing with MarginPolish and HELEN) to around 36 (Supplementary Table 24). Notably, attempting to use Pilon polishing on the raw Shasta assembly resulted in much poorer results (QV = 24).

Figure 3.5d and Supplementary Table 25 describe average runtimes and costs for the methods (see Methods). MarginPolish and HELEN cost a combined US$107

and took 29 h of wall-clock time on average, per sample. In comparison Racon and Medaka cost US$621 and took 142 wall-clock hours on average, per sample. To assess single-region performance we additionally ran the two polishing workflows on a single contig (roughly 1% of the assembly size), MarginPolish/HELEN was three times faster than Racon (1×)/Medaka (Supplementary Table 26).

Long-read assemblies contain nearly all human coding genes

To evaluate the accuracy and completeness of an assembled transcriptome we ran the Comparative Annotation Toolkit[98], which can annotate a genome assembly using the human GENCODE[99] reference human gene set (Table 3.1, Methods and Supplementary Tables 27–30).

Table 3.1: CAT transcriptome analysis of human protein coding genes for HG00733 and CHM13.

Sample	Assembler	Polisher	Genes Found %	Missing Genes	Complete Genes %
HG00733	Canu	HELEN	99.741	51	67.038
	Flye	HELEN	99.405	117	71.768
	Wtdbg2	HELEN	97.429	506	66.143
	Shasta	HELEN	99.228	152	68.069
	Shasta	Medaka	99.141	169	66.27
CHM13	Shasta	HELEN	99.111	175	74.202
	Shasta	Medaka	99.035	190	73.836

For the HG00733 and CHM13 samples we found that Shasta assemblies polished with MarginPolish and HELEN contained nearly all human protein coding genes,

having, respectively, an identified ortholog for 99.23% (152 missing) and 99.11% (175 missing) of these genes. Using the restrictive definition that a coding gene is complete in the assembly only if it is assembled across its full length, contains no frameshifts and retains the original intron–exon structure, we found that 68.07% and 74.20% of genes, respectively, were complete in the HG00733 and CHM13 assemblies. Polishing the Shasta assemblies alternatively with the Racon–Medaka pipeline achieved similar but uniformly less complete results.

Comparing the MarginPolish and HELEN polished assemblies for HG00733 generated with Flye, Canu and Wtdbg2 to the similarly polished Shasta assembly we found that Canu had the fewest missing genes (just 51), but that Flye, followed by Shasta, had the most complete genes. Wtdbg2 was clearly an outlier, with notably larger numbers of missing genes (506). For comparison we additionally ran BUSCO[100] using the eukaryote set of orthologs on each assembly, a smaller set of 303 expected single-copy genes (Supplementary Tables 31 and 32). We find comparable performance between the assemblies, with small differences largely recapitulating the pattern observed by the larger CAT analysis.

Comparison of Shasta and PacBio HiFi assemblies

We compared the CHM13 Shasta assembly polished using MarginPolish and HELEN with the recently released Canu assembly of CHM13 using PacBio HiFi reads[101]. HiFi reads are based on circular consensus sequencing technology that delivers substantially lower error rates. The HiFi assembly has a lower NG50 (29.0 versus 41.0 megabase

(Mb)) than the Shasta assembly (Supplementary Fig. 6). Consistent with our other comparisons to Canu, the Shasta assembly also contains a much lower disagreement count relative to GRCh38 (1073) than the Canu-based HiFi assembly (8,469), a difference that remains after looking only at disagreements within the intersection of the assemblies (380 versus 594). The assemblies have an almost equal NGAx (˜20.0 Mb), but the Shasta assembly covers a smaller fraction of GRCh38 (95.28 versus 97.03%) (Supplementary Fig. 7 and Supplementary Table 33). Predictably, the HiFi assembly has a higher QV than the polished Shasta assembly (QV = 41 versus QV = 32).

Scaffolding to near chromosome scale

To achieve chromosome length sequences, we scaffolded all of the polished Shasta assemblies with HiC proximity-ligation data using HiRise [57] (see Methods and Fig. 3.6a). On average, 891 joins were made per assembly. This increased the scaffold NG50 values to near chromosome scale, with a median of 129.96 Mb, as shown in Fig. 3.6a, with additional assembly metrics in Supplementary Table 36. Proximity-ligation data can also be used to detect misjoins in assemblies. In all 11 Shasta assemblies, no breaks to existing contigs were made while running HiRise to detect potential misjoins. Aligning HG00733 to GRCh38, we find no notable rearrangements and all chromosomes are spanned by one or a few contigs (Fig. 3.6b), with the exception of chrY, which is absent because HG00733 is female. Similar results were observed for HG002 (Supplementary Fig. 8).

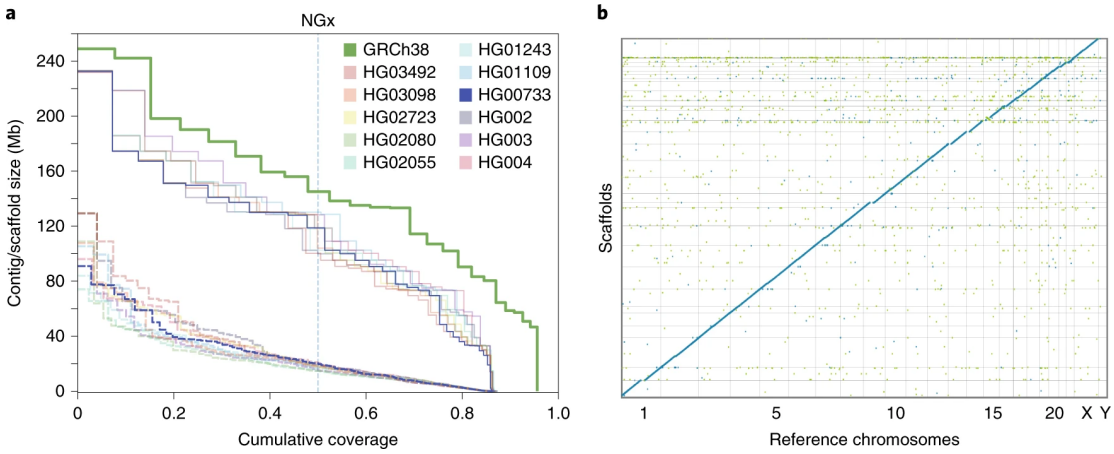

Figure 3.6: **HiRise scaffolding for 11 genomes.** a, NGx plots for each of the 11 genomes, before (dashed) and after (solid) scaffolding with HiC sequencing reads, GRCh38 minus alternate sequences is shown for comparison. b, Dot plot showing alignments between the scaffolded HG00733 Shasta assembly and GRCh38 chromosome scaffolds. Blue indicates forward aligning segments, green indicates reverse, with both indicating unique alignments.

Discussion

With sequencing efficiency for long reads improving, computational considerations are paramount in determining overall time, cost and quality. Simply put, large genome de novo assembly will not become ubiquitous if the requirements are weeks of

assembly time on large computational clusters. We present three new methods that provide a pipeline for the rapid assembly of long nanopore reads. Shasta can produce a draft human assembly in around 6 hours and US$70 using widely available commercial cloud nodes. This cost and turnaround time is much more amenable to rapid prototyping and parameter exploration than even the fastest competing method (Wtdbg2), which was on average 7.5 times slower and 3.7 times more expensive.

The combination of the Shasta assembler and nanopore long-read sequences produced using the PromethION sequencer realizes substantial improvements in throughput; we completed all 2.3 Tb of nanopore data collection in 9 d, running up to 15 flow cells simultaneously.

In terms of assembly, we obtained an average NG50 of 18.5 Mb for the 11 genomes, roughly three times higher than for the first nanopore-sequenced human genome, and comparable with the best achieved by alternative technologies[71, 102]. We found the addition of HiC sequencing for scaffolding necessary to achieve chromosome scale assemblies. However, our results are consistent with previous modeling based on the size and distribution of large repeats in the human genome, which predicts that an assembly based on 30 times coverage of such reads of > 100 kb would approach the continuity of complete human chromosomes[33, 84].

Relative to alternate long-read and linked-read sequencing, the read identity of nanopore reads is lower, however, improving over time[70, 33]. We observe modal read identity of 92.5%, resulting in better than QV = 30 base quality for haploid polished assembly from nanopore reads alone. The accurate resolution of highly repetitive and

recently duplicated sequence will depend on long-read polishing, because short reads are generally not uniquely mappable. Further polishing using complementary data types, including PacBio HiFi reads49 and 10X Chromium50, will likely prove useful in achieving QV 40+ assemblies.

Shasta produces a notably more conservative assembly than competing tools, trading greater correctness for contiguity and total produced sequence. For example, the ratio of total length to aligned length is relatively constant for all other assemblers, where approximately 1.6% of sequence produced does not align across the three evaluated samples. In contrast, on average just 0.38% of Shasta's sequence does not align to GRCh38, representing a more than four times reduction in unaligned sequence. Additionally, we note substantially lower disagreement counts, resulting in much smaller differences between the raw NGx and corrected NGAx values. Shasta also produces substantially more base-level accurate assemblies than the other competing tools. MarginPolish and HELEN provide a consistent improvement of base quality over all tested assemblers, with more accurate results than the current state-of-the-art long-read polishing workflow.

We assembled and compared haploid, trio-binned and diploid samples. Trio-binned samples show great promise for haplotype assembly, for example contiguously assembling an MHC haplogroup, but the halving of effective coverage resulted in ultimately less contiguous human assemblies with higher base-error rates than the related, chimeric diploid assembly. This can potentially be rectified by merging the haplotype assemblies to produce a pseudo-haplotype or increasing sequencing coverage. Indeed,

the improvements in contiguity and base accuracy in CHM13 over the diploid samples illustrate what can be achieved with higher coverage of a haploid sample. We believe that one of the most promising directions for the assembly of diploid samples is the integration of phasing into the assembly algorithm itself, as pioneered by others[74, 103, 104]. We anticipate that the new tools we have described here are suited for this next step: the Shasta framework is well placed for producing phased assemblies over structural variants, MarginPolish is built off of infrastructure designed to phase long reads2 and the HELEN model could be improved to include haplotagged features for the identification of heterozygous sites.

Connected together, the tools we present enabled a polished assembly to be produced in around 24 h and for roughly US$180, against the fastest comparable combination of Wtdbg2, Racon and Medaka that costs 5.3 times more and is 4.3 times slower while producing measurably worse results in terms of disagreements, contiguity and base-level accuracy. Substantial further parallelism of polishing, the main time drain in our current pipeline, is easily possible.

We are working toward the goal of having a half-day turnaround of our complete computational pipeline. With real-time basecalling, a DNA-to-de novo assembly could conceivably be achieved in less than 96 h. Such speed would enable screening of human genomes for abnormalities in difficult-to-sequence regions.

Methods

A selection of the methods from the Shasta paper are included here. The rest can be found in the Appendix.

Analysis methods

QUAST / BUSCO

To quantify contiguity, we primarily depended on the tool QUAST [85]. QUAST identifies misassemblies as major rearrangement events in the assembly relative to the reference. We use the phrase *disagreement* in our analysis, as we find "misassembly" inappropriate considering potentially true structural variation. For our assemblies, we quantified all contiguity stats against GRCh38, using autosomes plus chromosomes X and Y only. We report the total disagreements given that their relevant "size" descriptor was greater than 1 Kb, as is the default behavior in QUAST. QUAST provides other contiguity statistics in addition to disagreement count, notably total length and total aligned length as reported in Figure 3.3d. To determine total aligned length (and unaligned length), QUAST performs collinear chaining on each assembled contig to find the best set of non-overlapping alignments spanning the contig. This process contributes to QUAST's disagreement determination. We consider unaligned sequence to be the portions of the assembled contigs which are not part of this best set of non-overlapping alignments. All statistics are recorded in Supplementary Table D.1. For all QUAST analyses, we used the flags `min-identity 80` and `fragmented`.

QUAST also produces an NGAx plot (similar to an NGx plot) which shows the aligned segment size distribution of the assembly after accounting for disagreements and unalignable regions. The intermediate segment lengths that would allow NGAx plots to be reproduced across multiple samples on the same axis (as is shown in Figure 3.3b) are not stored, so we created a GitHub fork of QUAST to store this data during execution: `https://github.com/rlorigro/quast`. Finally, the assemblies and the output of QUAST were parsed to generate figures with an NGx visualization script, `ngx_plot.py`, found at `github.com/rlorigro/nanopore_assembly_and_polishing_assessment/`. For NGx and NGAx plots, a total genome size of 3.23Gb was used to calculate cumulative coverages.

BUSCO [100] is a tool which quantifies the number of Benchmarking Universal Single-Copy Orthologs present in an assembly. We ran BUSCO via the option within QUAST, comparing against the *eukaryota* set of orthologs from OrthoDB v9.

Disagreement assessments

To analyze the QUAST-reported disagreements for different regions of the genome, we gathered the known segmental duplication (SD) regions [105], centromeric regions for GRCh38, and known regions in GRCh38 with structural variation for HG002 from GIAB [88]. We used a Python script `quast_sv_extractor.py` that compares each reported disagreement of QUAST to the SD, SV and centromeric regions and discounts any disagreement that overlaps with these regions. The `quast_sv_extractor.py` script can be found at `https://github.com/kishwarshafin/helen/blob/master/modules/`

`python/helper/`.

The segmental duplication regions of GRCh38, "ucsc.collapsed.sorted.segdups", can be downloaded from `https://github.com/mvollger/segDupPlots/`.

The defined centromeric regions of GRCh38 for all chromosomes are used from the available summary at `https://www.ncbi.nlm.nih.gov/grc/human`.

For GIAB HG002, known SVs for GRCh38 are in "NIST_SVs_Integration_v0.6/" in `ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/`. We used the `Tier1+2` bed file available at the GIAB ftp site.

We further exclude SV enriched regions like centromeres, secondary constriction regions, acrocentric arms, large tandem repeat arrays, segmental duplications and the Y chromosome plus 10 kbp on either side of them. The file is available at:

`https://github.com/kishwarshafin/helen/blob/master/masked_regions/GRCh38_masked_regions.bed`

To analyse disagreements within the intersection of the assembled sequences we performed the following analysis. For each assembly we used `minimap2` and `samtools` to create regions of unique alignment to GRCh38. For minimap2 we used the options `--secondary=no -a --eqx -Y -x asm20 -m 10000 -z 10000,50 -r 50000 -O 5,56 -E 4,1 -B 5 --end-bonus=100` . We fed these alignments into `samtools view` with options `-F 260 -u -` and then `samtools sort` with option `-m`. We then scanned 100 basepair windows of GRCh38 to find windows where all assemblies for the given sample were aligned with a 1-1 mapping to GRCh38. We then report the sum of

disagreements across these windows. The script for this analysis is here: `https://github.com/mvollger/consensus_regions`.

Trio-binning

We performed trio-binning on two samples HG002 and HG00733 [91]. For HG00733, we obtained the parental read sample accessions (HG00731, HG00732) from 1000 genome database. Then we counted k-mers with `meryl` to create maternal and paternal k-mer sets. Based on manual examination of the k-mer count histograms to determine an appropriate threshold, we excluded k-mers occurring less than 6 times for maternal set and 5 times for paternal set. We subtracted the paternal set from the maternal set to get k-mers unique to the maternal sample and similarly derived unique paternal k-mer set. Then for each read, we counted the number of occurrences of unique maternal and paternal k-mers and classified the read based on the highest occurrence count. During classification, we avoided normalization by k-mer set size. This resulted in 35.2x maternal, 37.3x paternal, and 5.6x unclassified for HG00733. For HG002, we used the Illumina data for the parental samples (HG003, HG004) from GIAB project [40]. We counted k-mers using `meryl` and derived maternal paternal sets using the same protocol. We filtered k-mers that occur less than 25 times in both maternal and paternal sets. The classification resulted in 24x maternal, 23x paternal, and 3.5x unknown. The commands and data source are detailed in the Supplementary Notes.

Transcript analysis with comparative annotation toolkit

We ran the Comparative Annotation Toolkit [98] to annotate the polished assemblies in order to analyze how well Shasta assembles transcripts and genes. Each assembly was individually aligned to the GRCh38 reference assembly using Cactus [106] to create the input alignment to CAT. The GENCODE [107] V30 annotation was used as the input gene set. CAT was run in the transMap mode only, without Augustus refinement, since the goal was only to evaluate the quality of the projected transcripts. All transcripts on chromosome Y were excluded from the analysis since some samples lacked a Y chromosome.

Run-Length Confusion Matrix

To generate run-length confusion matrices from reads and assemblies, we run-length encoded (RLE) the assembly/read sequences and reference sequences using a purpose-built python script, `measure_runlength_distribution_from_fasta.py`. The script requires a reference and sequence file, and can be found in the GitHub repo `https://github.com/rlorigro/runlength_analysis/`. The RLE nucleotides were aligned to the RLE reference nucleotides with `minimap2`. As RLE sequences cannot have identical adjacent nucleotides, the number of unique k-mers is diminished with respect to standard sequences. As `minimap2` uses empirically determined sizes for seed k-mers, we used a k-mer size of 19 to approximately match the frequency of the default size (15) used by the presets for standard sequences. For alignment of reads and assemblies we used the `map-ont` and `asm20` presets respectively.

By iterating through the alignments, each match position in the cigar string (mismatched nucleotides are discarded) was used to find a pair of lengths (x, y) such that x is a predicted length and y is the true (reference) length. For each pair, we updated a matrix which contains the frequency of every possible pairing of prediction vs truth, from length 1bp to 50bp. Finally, this matrix is normalized by dividing each element by the sum of the observations for its true run length, $\sum_{i=1}^{50}(x_i, y)$, and plotted as a heatmap. Each value represents the probability of predicting a length for a given true length.

Runtime and Cost Analysis

Our runtime analysis was generated with multiple methods detailing the amount of time the processes took to complete. These methods include the unix command `time` and a home-grown resource tracking script which can be found in the `https://github.com/rlorigro/TaskManager` repository. We note that the assembly and polishing methods have different resource requirements, and do not all fully utilize available CPUs, GPUs, and memory over the program's execution. As such, we report runtimes using wall clock time and the number of CPUs the application was configured to use, but do not convert to CPU hours. Costs reported in the figures are the product of the runtime and AWS instance price. Because portions of some applications do not fully utilize CPUs, cost could potentially be reduced by running on a smaller instance which would be fully utilized, and runtime could be reduced by running on a larger instance which can be fully utilized for some portion of execution. We particularly note the long

runtime of Medaka and found that for most of the total runtime, only a single CPU was used. Lastly, we note that data transfer times are not reported in runtimes. Some of the data required or generated exceeds hundreds of gigabytes, which could be potentially significant in relation to the runtime of the process. Notably, the images generated by MarginPolish and consumed by HELEN were often greater than 500 GB in total.

All recorded runtimes are reported in the supplement. For Shasta, times were recorded to the tenth of the hour. All other runtimes were recorded to the minute. All runtimes reported in figures were run on the Amazon Web Services cloud platform (AWS).

Shasta runtime reported in Fig. 3.3f was determined by averaging across all 12 samples. Wtdbg2 runtime was determined by summing runtimes for wtdbg2 and wtpoa-cns and averaging across the HG00733, HG002, and CHM13 runs. Flye runtime was determined by averaging across the HG00733, HG002, and CHM13 runs, which were performed on multiple instance types (x1.16xlarge and x1.32xlarge). We calculated the total cost and runtime for each run and averaged these amounts; no attempt to convert these to a single instance type was performed. Precise Canu runtimes are not reported, as they were run on the NIH Biowulf cluster. Each run was restricted to nodes with 28 cores (56 hyperthreads) (2x2680v4 or 2x2695v3 Intel CPUs) and 248GB of RAM or 16 cores (32 hyperthreads) (2x2650v2 Intel CPUs) and 121GB of RAM. Full details of the cluster are available at `https://hpc.nih.gov`. The runs took between 219 and 223 thousand CPU hours (4-5 wall-clock days). No single job used more than 80GB of RAM/12 CPUs. We find the r5.4xlarge ($1.008 per hour) to be the cheapest

AWS instance type possible considering this resource usage, which puts estimated cost between $18,000 and $19,000 per genome.

For MarginPolish, we recorded all runtimes, but used various thread counts that did not always fully utilize the instance's CPUs. The runtime reported in the figure was generated by averaging across 8 of the 12 samples, selecting runs that used 70 CPUs (of the 72 available on the instance). The samples this was true for were GM24385, HG03492, HG01109, HG02055, HG02080, HG01243, HG03098, and CHM13. Runtimes for read alignments used by MarginPolish were not recorded. Because MarginPolish requires an aligned BAM, we found it unfair to not report this time in the figure as it is a required step in the workflows for MarginPolish, Racon, and Medaka. As a proxy for the unrecorded read alignment time used to generate BAMs for MarginPolish, we added the average alignment time recorded while aligning reads in preparation for Medaka runs. We note that the alignment for MarginPolish was done by piping output from `minimap2` directly into `samtools sort`, and piping this into `samtools view` to filter for primary and supplementary reads. Alignment for Medaka was done using `mini_align`, which is a wrapper for `minimap2` bundled in Medaka that simultaneously sorts output.

Reported HELEN runs were performed on GCP except for HG03098, but on instances that match the AWS instance type `p2.8xlarge` in both CPU count and GPU (NVIDIA Tesla P100). As such, the differences in runtime between the platforms should be negligible, and we have calculated cost based on the AWS instance price for consistency. The reported runtime is the sum of time taken by `call_consensus.py` and

`stitch.py`. Unannotated runs were performed on UCSC hardware.

Racon runtimes reflect the sum of four series of read alignment and polishing. The time reported in the figure is the average of the runtime of this process run on the Shasta assembly for HG00733, HG002, and CHM13.

Medaka runtime was determined by averaging across the HG00733, HG002, and CHM13 runs after running Racon 4× on the Shasta assembly. We again note that this application in particular did not fully utilize the CPUs for most of the execution, and in the case of HG00733 appeared to hang and was restarted. The plot includes the average runtime from read alignment using `minialign`; this is separated in the tables in the supplementary results. We ran Medaka on an `x1.16xlarge` instance, which had more memory than was necessary. When determining cost, we chose to price the run based on the cheapest AWS instance type that we could have used accounting for configured CPU count and peak memory usage (`c5n.18xlarge`). This instance could have supported 8 more concurrent threads, but as the application did not fully utilize the CPUs we find this to be a fair representation.

Chapter 4

Gene Annotation

Background

Comparative genomics at scale

As DNA sequencing and assembly has become increasingly accessible in recent years, hundreds of genomes have been published, arising from projects such as Genome 10k [108], the Vertebrate Genome Project, and the 200 Mammals Project, with many more hundreds to thousands expected in the next decade [109]. This allows for the studies of comparative genomics at scales not previously possible. However, many of these newer genomes were created using next-generation sequencing types, which means that the genome assemblies are relatively discontiguous. They also frequently contain other types of errors, like misjoins between contigs. Many comparative genomics analyses require genomes which are both high quality at a base-level and highly contiguous [110], so we are not able to fully take advantage of the enormous number of published

genome assemblies at this time. Fortunately, genome assembly techniques are quickly improving (as seen in the previous chapter), and the number of species with high-quality, contiguous genomes is quickly increasing. We can use these new assemblies to improve alignments between genomes and improve gene annotations for these species. Alongside this, algorithms for comparative genomics analyses have improved to take advantage of the higher quality input data.

Whole genome alignment

The first step in many comparative genomics analyses is aligning entire genomes together to find homologous regions to compare in a process called whole-genome alignment. Ideally, we would be able to reconstruct the evolutionary history of each base pair in a set of sequences; however, the similarity of the base DNA sequences can be used as a proxy for the level of homology of the sequences. At its basis, a whole-genome alignment is not any different than shorter alignments, but the size of the input sequences can lead to huge computational complexity, and extra consideration needs to be given to large modifications to the input sequences such as structural rearrangements.

Having high-quality, long alignments is necessary for accurate comparison, and the higher quality the input genomes are, the more accurate this alignment can be. One tool for creating these whole-genome alignments is Cactus [111], which was later extended to progressiveCactus [109]. Cactus uses a cactus graph to find high-confidence anchors for the alignment, by starting with small local alignments to first naively combine into a multiple alignment and then filter. The extension progressiveCactus can

scale to hundreds of genomes by using a phylogenetic tree to guide the alignments and break the alignment into subproblems.

Gene annotation

Annotating the location of the functional elements within a genome is often the next step in a comparative genomics analysis, as one can imagine wanting to compare the elements of the genome that have identifiable functions between species. I focused on gene annotation specifically in my dissertation, which is the process of determining the locations of genes and other coding regions in the genome. Because much of the genome evolution we are interested in involves the functional coding part of the genome, having high quality gene annotations is essential.

There are two main strategies for creating a gene annotation for a genome assembly: *ab initio* prediction, where the gene structures are predicted using computational models, and alignment-based prediction, where a set of existing sequences are mapped onto an assembly to find the locations of transcripts. Of course, the two approaches can also be combined. Many annotation sets currently are produced by either NCBI (RefSeq), or Ensembl (Ensembl Gene Build). These institutions have many resources, but outsourcing the gene annotation portions of pipelines is increasingly unnecessary as annotation can be performed in-house.

One software option that can perform both alignment-based annotations and *ab initio* predictions is the Comparative Annotation Toolkit (CAT), which I used (and extended) for much of the remaining part of my dissertation. Another option is MAKER2,

which can also incorporate both types of information, but is technically challenging to run, does not track orthology relationships and requires additional gene modeling processing, and requires assembly of extrinsic transcripts into a transcriptome in order to be used.

Comparative Annotation Toolkit

The Comparative Annotation Toolkit (CAT) [98] attempts to integrate multiple existing annotation methods, as described above. It was largely developed by Ian Fiddes for his dissertation while he was at UC Santa Cruz. CAT can take a reference-quality annotation of a genome (GFF3) and alignment of this genome and others (HAL) and produce an annotation on all target genomes as output. CAT can annotate multiple genomes simultaneously and identify orthology relationships due to its use of alignments from progressiveCactus, which are not reference-based and can include duplications. CAT can also incorporate additional data such as RNA-sequencing reads to improve annotations and identify novel genes. It can run many parameterizations of AUGUSTUS and combine all of those predictions when forming a final consensus gene set for each genome within the alignment.

CAT (and Cactus) have been used in combination in high-profile projects, such as the Great Apes project [112], and the next few chapters in this dissertation describe its use in primate assembly papers for the rhesus macaque and the bonobo, the Telomere-to-Telomere Consortium, and the Human Pangenome Reference Consortium.

CAT pipeline

The pipeline for CAT can be viewed in Figure 4.1.

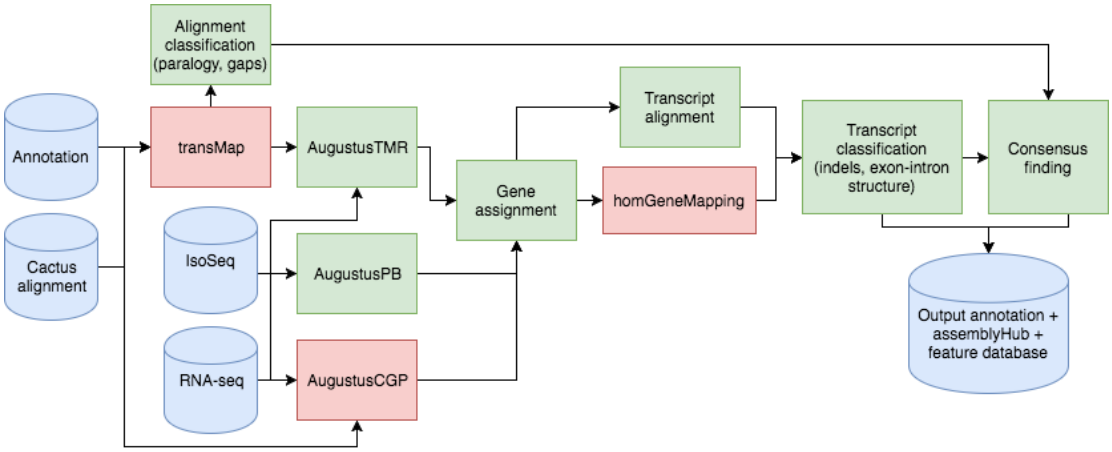

Figure 4.1: **Pipeline for Comparative Annotation Toolkit.**

The first step of the pipeline is the `transMap` module, which runs the utility `pslMap` in order to project transcript annotations from the reference genome onto a target genome, using alignments between the reference and target genome represented in `.chain` format. `transMapPslToGenePred` converts the transcript projections to a gene model, which keeps track of frame information and can fill in coding and noncoding gaps. After the initial transMap projection, the alignments are filtered down. This is the main step where paralog resolution is involved, so that alignments are filtered down to only include the most likely ortholog.

CAT can then be run with multiple parameterizations of AUGUSTUS: AugustusTM, AugustusTMR, AugustusCGP, and AugustusPB. The output of each of the output modes of Augustus is combined with the output of TransMap in a final consensus building step. AugustusTM/TMR are used to refine specific isoforms of transcripts, to do things like fixing regions where alignments dropped exons or introduced small gaps, or where splice sites may have shifted. The other two modes of Augustus, AugustusCGP and AugustusPB, are used to introduce novel isoforms of known genes, or entirely novel genes. AugustusCGP uses the knowledge of the phylogenetic tree to predict missing genes and transcripts, whereas AugustusPB uses IsoSeq data to predict novel genes and isoforms. AugustusPB in particular is able to predict genes in regions that were unable to align to other genomes.

Finally, CAT is able to incorporate additional information from extrinsic reference annotations. This can be the information from a well-studied reference annotation such as those from GENCODE, or it can be from a *de novo* constructed annotation from external reference data, such as using StringTie2 [113] to create an annotation from external Iso-Seq data. For any mode that may introduce a putatively novel transcript or gene, the assignment of a potential parent gene (or transcript) will be resolved during the parent gene assignment stage. First, putatively novel transcripts will be compared to the projections from transMap. If it can be confidently assigned to a gene, it will do so. If it were confidently assigned to a transcript with an orthologous projection, it will be evaluated to be a novel isoform of the gene. If the gene was later filtered out during paralog resolution, it will be considered as a potential candidate for a novel

gene as a potential paralog. If the candidate novel gene does not overlap with any prior predictions, it is the considered to be a putative novel gene.

The quality of the input genome alignments to CAT is crucial to its success. Particularly, having high quality outgroups will improve the resolution of paralogs and rearrangements. The assemblies themselves also need to be of high quality. Discontiguous assemblies can lead to poor alignments and annotated functional features being split between contigs. Indel errors, present particularly in long read assemblies, can result in low-quality gene models with many frameshift and nonsense mutations.

CAT software development

Though I did not conceive of or originally implement the Comparative Annotation Toolkit (CAT) [98], I contributed to some of its software development.

Running with chain files as input

Previously, CAT was only able to be run using an input HAL file as the source of alignments. I extended CAT to be able to use input *.chain* files as input. These files also describe how one region can be lifted over to another region, and are in fact the files created from the HAL file upon with `pslMap` is run during the `transMap` module of CAT. These chain files are also added to the automatic AssemblyHub generation process so they can be visualized within the UCSC Genome Browser.

Visualizing synteny files

CAT can now run halSynteny in order to automatically produce synteny files that can be visualized within the AssemblyHub.

Paralog resolution fixes

CAT (and many other gene annotation software programs) struggle with the resolution of paralogous genes. I added some additional fixes to improve paralog resolution within CAT.

Some of the difficulties arise from CAT trying to map each paralog of a gene to its single best locus. However, if overlapping genes are allowed, this leads to overlapping genes at the same locus, and if not, then the additional copy is not annotated anywhere, even if there is a second locus that would be a perfectly acceptable locus for it to be placed. In order to resolve this, I added some additional steps to paralog resolution. First, I consider the other genes in the same clusters and their scores and other possible locations. Also, I add penalties for overlapping genes with different IDs to the scores for each gene.

Speed improvements

Parent gene assignment is the step of CAT that assigns source genes to *de novo* gene predictions from other modules within the pipeline. The current implementation runs in $O(n^2)$ time, as it makes many unnecessary comparisons between each novel prediction and every source gene. Switching the implementation to use clusterGenes

should be much faster, as after transcripts are clustered to involve only transcripts that overlap each other, clustering can be performed within each cluster much more quickly.

I implemented this within CAT, and it is much faster. For a whole-genome sized run, the runtime is reduced from 12h 38m to only 36 minutes - a 21 time speedup!

Limitations and future directions

CAT is still limited by its reliance on the alignments between the reference genome and the genome being annotated.

If we relax the requirement that gene annotations come from the whole-genome alignments between the reference genome and target genome, we can perform additional alignments of reference transcripts to the target assembly itself in order to try to rescue some of the previously unmapped reference transcripts.

Another new feature would be introducing the ability to identify additional copies of genes in the target genome. These would be genes from families that have expanded in the target genome relative to the reference. The information needed to do this is essentially already there, as transcripts record additional loci where they also align sufficiently well. Then, there would need to be a step added to resolve these mappings (amongst all of the other genes in the family) and output these putative novel duplications in the final output files.

Gene annotation of primates

Background

Comparative genomics analyses of primates are of particular interest because the study of primates can elucidate a lot about human evolution. A phylogenetic tree of primates can be seen in Figure 4.2. Primate genomes, including humans, can be challenging to assemble because they are highly repetitive, with repetitive elements making up approximately 50% of their genomes [1], with large differences in species-specific insertions. Segmental duplications are an important factor which differentiate primate genomes, and are challenging to assemble even with long reads. Certain gene families, such as zinc-finger transcription factors, have many paralogs within each primate genome, and have rapidly evolved between primates with many gains and losses across branches.

Up until now, analysis of such repeat content has been difficult or impossible. Fortunately, more and more high quality genomes for a wide variety of primates are becoming available. New high-quality primate genomes have already shown improvement over prior versions in ways such as increased mapping identity and coverage of RNA-sequencing reads, and improved identification of structural variations [112]. We can continue to use these high quality assemblies to study the difficult regions, such as repeat families and pseudogenized genes, in primate genomes.

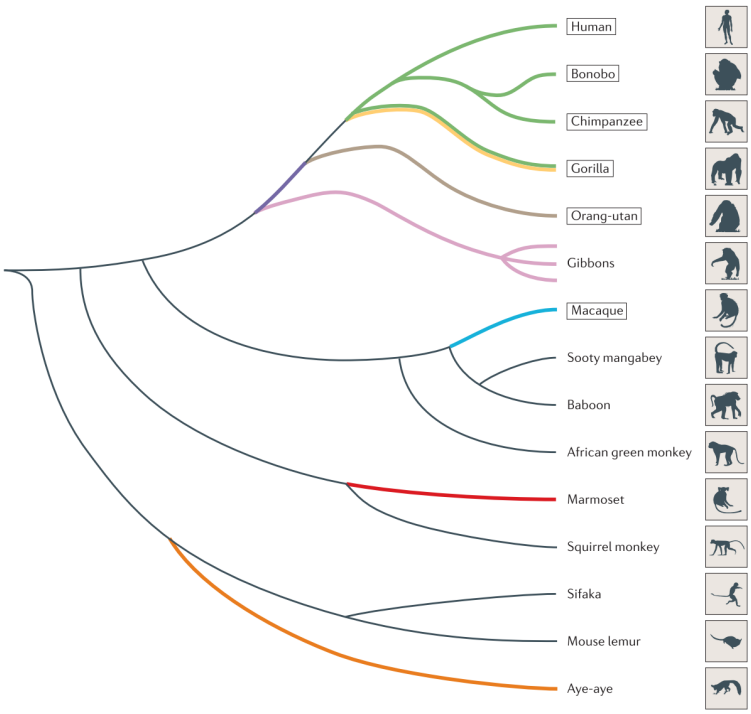

Figure 4.2: **Primate phylogenetic tree.** Taken from Rogers and Gibbs 2014 [1]. Evolutionary relationships among primates are shown.

Contributions for primate gene annotations

The next goal for my dissertation was to use high-quality long read primate and human assemblies to gain an understanding of gene gain and loss during the evolution of primates. New, higher quality primate assemblies have been assembled, using PacBio CLR reads to make the initial assemblies. These assemblies are much more contiguous than previous assemblies, while maintaining a high quality. We then aligned these assemblies together, and used the alignment to run CAT. The next sections of this

chapter present the results of performing gene annotation and analysis of the results for two primate species of those with new assemblies, the rhesus macaque and the bonobo. I was a coauthor on the papers that described the new assemblies and analysis, which were called "Sequence diversity analyses of an improved rhesus macaque genome enhance its biomedical utility", published in Science in December 2020 [114], and "A high-quality bonobo genome refines the analysis of hominid evolution", published in Nature in June 2021 [115].

Rhesus Macaque

Background

The rhesus macaque (*Macaca mulatta*) is the most widely studied nonhuman primate in biomedical research. Even though it is more distantly related to humans than some other primates, particularly the great apes (chimpanzees, bonobos, gorillas, and orangutans), rhesus macaque models have been used in the study of infectious diseases including Ebola and HIV/AIDS, and neurodevelopmental diseases such as autism [116, 117, 118]. The differences between human and macaque genomes are important to understand when macaques are used as a model for so many human diseases, so an increase in the quality of the rhesus macaque genome would enable a more thorough analysis of these genetic differences.

mmul10 assembly

The most recent version of the rhesus macaque genome, `mmul10`, has a contig N50 of 46 Mbp, a 120-fold increase over the prior version, `mmul8`. It was annotated using 6.5 million full-length cDNAs. The newest version of the rhesus macaque genome showed some improvement over the previous version in terms of transcript and IsoSeq alignments, as shown in Figure 4.3.

The rhesus macaque assembly was incorporated into an alignment with several other primate genomes (human, bonobo, chimp, gorilla, orangutan, gibbon, marmoset, and owl monkey). We then used this alignment to run the Comparative Annotation Toolkit (CAT) to annotate each of the genomes in this alignment. We incorporated the cDNA reads into the annotations in order to discover novel isoforms of genes. CAT was run using all AUGUSTUS modules: AugustusTM, AugustusTMR, AugustusPB, and AugustusCGP.

We used this improved assembly with annotations to discover novel lineage-specific genes and expanded gene families, and significantly improved understanding of gene content, isoform diversity, and repeat organization, elucidating differences along the rhesus lineage.

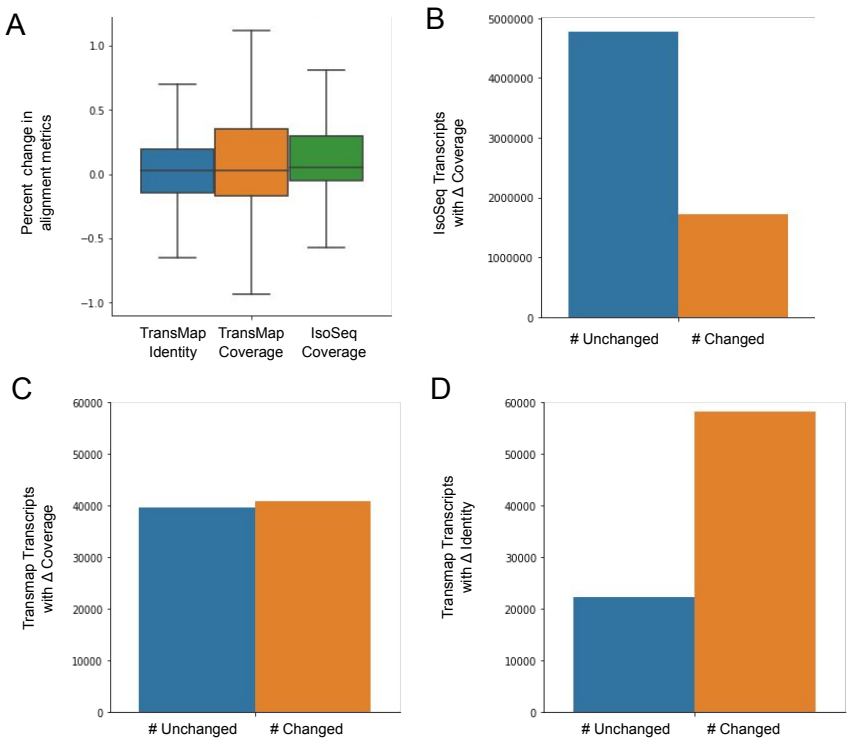

Figure 4.3: **TransMap and Iso-Seq mappability in Mmul 10 compared to Mmul 8.0.1.** Comparative Annotation Toolkit (CAT) was used to project transcripts from GRCh38 to Mmul 10 and Mmul 8.0.1. Alignment coverage and identity were compared for orthologous transcripts found in each assembly pair. Additionally, Iso-Seq transcripts were mapped to both Mmul 10 and Mmul 8.0.1. (A) The box plots show the percentage change in identity and coverage of the TransMap alignments (left, middle), and coverage of Iso-Seq alignments (right) between Mmul 10 and Mmul 8.0.1. Transcripts with unchanged metrics were omitted from the plot. (B) Number of Iso-Seq transcripts that had changes in coverage between the assemblies. (C,D) Number of TransMap transcripts that had changes in coverage and identity between Mmul 10 and Mmul 8.0.1.

Gene annotation on mmul10

I analyzed the CAT annotations of the 83,692 protein coding isoforms found on the `mmul10` assembly and noted relatively small numbers genes with problems such as frame-shifting indels (980) and genes split over multiple contigs (550). There are 827

genes with reduced copy number relative to humans. We compared these annotations to annotations made on the `mmul8.0.1` assembly to show that using the `mmul10` assembly improves the quality of the resulting annotated transcripts, visualized in Figure 4.3.

I also identified candidate novel genes, which we defined as genes which were in the rhesus macaque assembly, but not the human gene annotation, meaning not necessarily genes that arose only in the rhesus lineage. This work is presented in Figure 4.4. In summary, CAT predicted 2,880 novel transcripts that did not arise from any previously annotated transcript in the input human annotation. Many were short, but there were 261 with at least five exons. I found three transcripts that are homologous to the human CYP2C18 protein, and had abundant IsoSeq support across multiple tissues (4.4A). We further identified a set of 84 novel exons (exons that were not present in the human annotation). We found two novel exons in the tropoelastin gene (ELN) near the C-terminus of the protein which had strong IsoSeq support for alternatively spliced isoforms across different tissue times (4.4B). These exons are not found in great or lesser apes, but are shared to the base of the mammalian tree, showing that this is a loss of exons in the ape lineage.

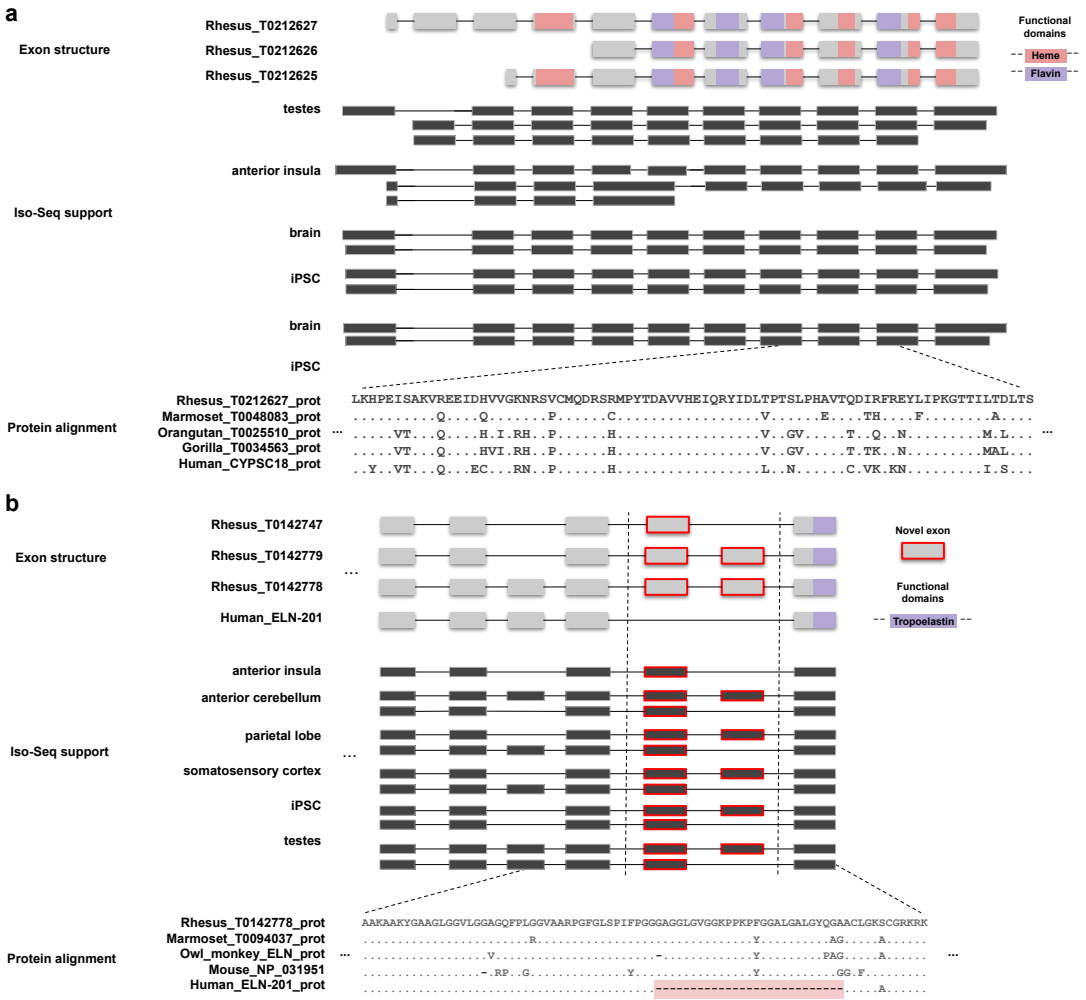

Figure 4.4: **Novel genes and gene models in the rhesus macaque.** (A) A novel gene model with homology to the cytochrome p450 protein family is predicted by the AugustusPB mode of the Comparative Annotation Toolkit (CAT). The gene structure and protein domain architecture of three isoforms are shown (top). The predictions arose from supporting Iso-Seq reads from five tissues (middle). Orthologous novel genes are also predicted in marmoset, orangutan, and gorilla assemblies; a protein alignment (bottom) of those genes along with a human CYP2C18 protein is shown. (B) Two macaque isoforms in ELN (tropoelastin) are predicted by the AugustusPB mode of CAT and are supported by macaque Iso-Seq data but differ significantly from human by two exons. The gene structure and functional domains for the last seven exons of this gene are shown (top), along with a comparison to a human transcript model. These two protein-encoding exons are also observed in marmoset, owl monkey, and mouse but not in apes, as a result of an ape-specific deletion (bottom) that changed the gene structure of tropoelastin.

Bonobo

Background

The primate alignment previously mentioned can be used for many additional analyses. The alignment included humans, great apes (bonobo, chimp, gorilla, and orangutan), lesser apes (gibbon), Old World monkeys (rhesus macaque), and New World monkeys (marmoset and owl monkey). The inclusion of all these different species allows us to study some of the major splits in the primate lineages, such as the split between New World and Old World monkeys. After the conclusion of the rhesus macaque genome analysis described in the previous section, there were plans to do the same for the latest bonobo assembly.

The bonobo genome (*Pan paniscus*) has been observed to contain a depletion of L1PA2 elements, making this particular species an interesting target for an in-depth analysis of KRAB zinc-finger genes and repeat family evolution. KRAB zinc finger genes have been involved in an evolutionary arms race with retrotransposable elements [119, 120]. However, KZNFs can be hard to distinguish from each other due to their self-similarity. The high-quality primate assemblies will allow us to narrow in on previously challenging gene families such as these.

panpan3 assembly

A new bonobo genome assembly was made from sequencing a female bonobo, Mhudiblu, to 74x coverage with PacBio reads. The new assembly, *panpan3*, has a contig

N50 of 16.58 Mb and includes an additional 74 Mb of sequence over the previous version (*panpan1.1?*), closing 99.5 percent of the 108,095 gaps in that assembly.

Gene annotation on panpan3

As previously described, we ran the Comparative Annotation Toolkit (CAT) on the large primate alignment that included this new bonobo assembly. We identified 20,478 protein-coding genes and 36,880 non-coding genes. I analyzed the rate of frameshift errors in the gene models and determined that only 0.5 percent of the genes (119) had insertions or deletions that caused a frameshift. Additionally, 38.4 percent of the protein-coding isoforms are more complete. 206 noncoding genes and 1,576 protein-coding genes are parts of gene families that either expanded or contracted compared to the human genome.

I also looked at the genes that had putatively novel exons discovered. There are 65 genes with novel exons that are supported by the cDNA evidence. One example of this is found in *ANAPC2*, where there is a novel exon found in the bonobo gene models but not in the chimpanzee. This is shown in Figure 4.5.

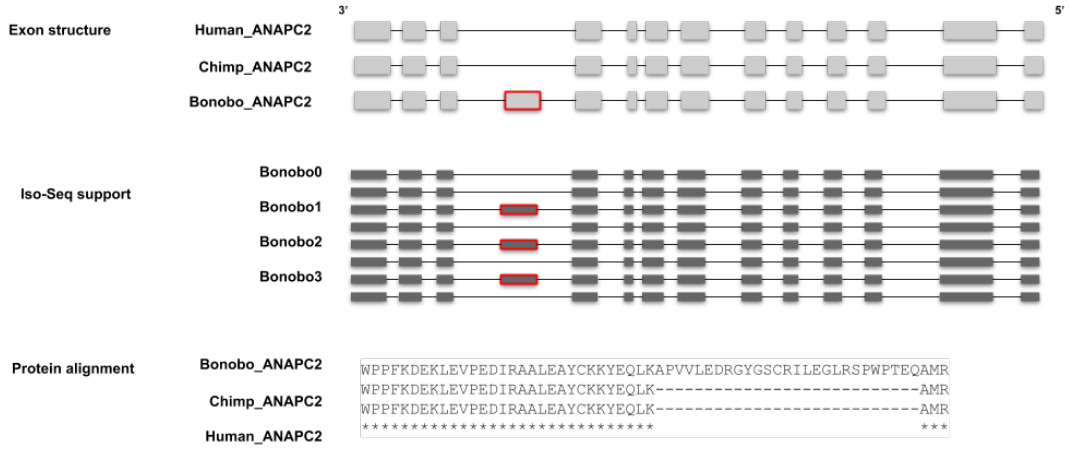

Figure 4.5: **A bonobo isoform of ANAPC2 contains a novel exon predicted by the AugustusPB mode of CAT, which is supported by bonobo Iso-Seq data from iPSC tissue.** The exon is not seen in the human or chimpanzee annotations. a, The exon structure of this gene is shown for human, chimpanzee, and bonobo. The novel exon is alternatively spliced, seen in one isoform of ANAPC2 in bonobo and not the other. b, A sample of Iso-Seq reads from the bonobo iPSC tissue shows support for alternative splicing in ANAPC2. c, A protein alignment of the gene is shown for human, chimpanzee, and bonobo.

Chapter 5

Gene annotation on a fully complete, telomere-to-telomere human assembly

Background

The human reference genome has gone through multiple rounds of improvements since the first human genome was completed at the end of the Human Genome Project in the year 2000. The initial project took over a decade and cost over $2.7 billion ($5 billion in 2022 dollars) [121, 122]. GRCh38 is the most recent version of the human genome, released by the Human Genome Reference Consortium in 2013 and most recently patched in 2019. It was constructed from thousands of individual bacterial artificial chromosomes (BACs). yeast artificial chromosomes (YACs), and fosmid clones, supplemented with whole-genome sequencing data. One downside of the GRCh38 reference is that it is a combination of the genomes of 20 anonymous volunteers (although

one individual represents approximately 70 percent of the sequence). This means that the sequences presented are not representative of what may occur in a haplotype of any single person, as they are a mosaic of many. Furthermore, this genome is not actually complete – it contains many "gaps" (995 gaps, which total approximately 151 Mb in size) where the DNA sequence at a locus is unknown. These gaps make up 8 percent of the human genome and contain regions that are biologically important. These gaps are located in regions of the genome that are challenging to sequence and assemble, namely in the centromeres and telomeres (and surrounding regions), segmental duplications, ampliconic gene arrays, and rDNA arrays. There are also other known errors in the GRCh38 assembly, such as a false duplication on chr21. These challenges can be traced back to the fact that the GRC assembly was made using bacterial artificial chromosomes (BACs), but these regions are still challenging to assemble using other types of DNA sequencing and automated assemblers.

Construction of the T2T assembly

The introduction of new types of DNA sequencing technologies allow for the possibility of constructing a truly complete human genome for the first time. PacBio HiFi reads are long (20kb length, error rate 0.1 percent) and highly accurate reads that can be used to create initial long, highly accurate contigs. Oxford Nanopore's ultralong reads (with read lengths greater than 100kb) can then be used to thread the contigs together to complete scaffolds of full chromosomes.

Contributions to the T2T publication

After the T2T assembly was created, I contributed to the work that was published in the T2T paper called "The complete sequence of a human genome"[123] by being one of the leads of the gene annotation team. The paper was published in Science in March 2022. I performed the gene annotation on the assembly and analyzed those results. I created figures and tables for the paper and contributed to writing the text for the relevant sections. The next parts of this chapter describe this work. The gene annotations produced were also used in the companion paper "Complete genomic and epigenetic maps of human centromeres" [124], also published in Science, and I am a coauthor on that manuscript as well.

T2T-CHM13 assembly construction and stats

The complete T2T CHM13 assembly contains telomere-to-telomere assemblies for each of the 22 chromosomes, chromosome X, and a mitochondrial chromosome. There are zero gaps. An additional 182 Mb of sequence was added that do not align to anywhere in the existing GRCh38 reference. The new sequences are primarily composed of centromeric satellites, segmental duplications, and ribosomal DNA (rDNA) arrays.

Gene annotation on the T2T-CHM13 assembly

Once the complete T2T-CHM13 genome was assembled, it needed to be annotated with the locations of genes. I used the Comparative Annotation Toolkit (CAT)

[98] to produce an initial gene annotation, lifting over GENCODE v35 gene sequences onto the T2T-CHM13 genome. First, I created an alignment of the T2T-CHM13 genome to GRCh38 was made using Cactus [109], using chimp (PanTro6) as an outgroup. I then assembled the Iso-Seq reads into a transcriptome using Stringtie2 [113] and use these as an external data source for the gene annotation. Next, I ran CAT using the resulting alignment file and the assembled transcriptome.

In addition to using CAT's annotations, Liftoff [125] was also run on the T2T-CHM13 assembly. This was a collaboration with Alaina Shumate from the Salzberg lab. In comparison to CAT, Liftoff does not depend on a whole-genome alignment of the genome to the reference to lift over the annotations. Instead, it lifts over each of the reference gene annotations to the genome of interest using minimap2. Therefore, Liftoff is capable of annotating genes in regions where there is no good alignment between GRCh38 and the T2T-CHM13 genome, whereas CAT is unable to. An additional feature of Liftoff is that it can identify additional copies of genes in the genome.

So, we combined the two gene sets from CAT and Liftoff to form a gene set that was more complete than either of them were on their own. We removed any genes from the CAT set that looked problematic (mainly paralogs that were annotated on top of each other), and added in any genes from Liftoff that did not overlap any of the CAT annotations.

In the final combined gene set (see Table 5.1), there were a total of 63,494 genes (19,969 protein-coding). In comparison to the GENCODE v35 annotation set on GRCh38, there are 3,604 new genes (140 protein-coding). There were 263 genes (63

protein-coding) that were unable to be lifted over to the T2T-CHM13 assembly. Some of these were expected, such as the genes located on the falsely duplicated section of chr21.

Statistics	GRCh38	T2T-CHM13	Difference (%)
Number of genes	60,090	63,494	+5.7
Protein coding	19,890	19,969	+0.4
Number of exclusive genes	263	3,604	
Protein coding	63	140	
Number of transcripts	228,597	233,615	+2.2
Protein coding	84,277	86,245	+2.3
Number of exclusive transcripts	1,708	6,693	
Protein coding	829	2,780	

Table 5.1: Comparison of GRCh38 and T2T-CHM13v1.1 human genome assemblies. GRCh38 summary statistics exclude "alts" (110 Mbp), patches (63 Mbp), and chromosome Y (58 Mbp). The number of exclusive genes or transcripts is as follows: for GRCh38, GENCODE genes and transcripts not found in CHM13; and for CHM13, extra putative paralogs that are not in GENCODE.

Gene annotation on the HG002 chrY assembly

Though the T2T-CHM13 genome was fully complete for the CHM13 cell line, there was no Y chromosome to be sequenced and assembled, as CHM13 did not have one. There was then an effort to produce a complete sequence of a Y chromosome. HG002 was selected for this, and the first complete Y chromosome assembly was made.

A similar process was used to annotate the HG002 chrY assembly. I made an alignment of the newly assembled chrY to GRCh38, using chimp as an outgroup. I then

used CAT to lift over the GENCODE v35 annotations to the new assembly, adding in the Iso-Seq data using the AugustusPB mode. Like before, the CAT annotations were combined with the Liftoff annotations. There was an additional step of manual curation to fix many of the ampliconic genes (e.g. the TSPY family) along the chromosome. The gene annotations for these genes were chosen based on protein sequence identity, and at any loci where the manual curation differed from that chosen by the CAT/Liftoff pipeline, the manually curated gene was inserted instead. The gene annotation results can be found in Table 5.2.

We identified 693 genes (107 of which were protein-coding), which was an 42 protein-coding genes compared to the chrY gene annotation on GRCh38.

		T2T-CHM13v2.0Y (HG002Y)	GRCh38Y	Change	
Assembly	Total num. bases	62,460,029	57,264,655	109.1%	5,195,374
	Assigned	62,460,029	57,227,415	109.1%	5,232,614
	Assigned, unlocalized	0	37,240		
	Num. gaps	0	56		
	Num. N-bases	0	30,812,366		53.8%
Annotation	Total num. genes	693	589	1.2	
	Excl.	110	0	na	
	Excl.	210	0	na	
	Protein-coding genes	107	66	1.6	
	Excl.	42	0	na	
	Protein-coding transcripts	493	372	1.3	
	Excl.	124	0	na	
Ampliconic gene copy numbers	BPY2	4 (3, 0)	4 (3, 0)	1.0	
	CDY	26 (4, 0)	26 (4, 0)	1.0	
	DAZ	4 (4, 0)	4 (4, 0)	1.0	
	HSFY	8 (2, 0)	8 (2, 0)	1.0	
	PRY	8 (2, 0)	8 (2, 0)	1.0	
	RBMY	34 (6, 4)	32 (6, 4)	1.1	
	TSPY	66 (46, 0)	25 (7, 0)	2.6	
	VCY	2 (2, 0)	2 (2, 0)	1.0	
	XKRY	8 (0, 2)	8 (0, 2)	1.0	
Haplogroup	Haplogroup abr.	J-L816 (J1a2b3a1)	R-L20 (R1b1a2a1a2b1a1)	na	
	Ancestry	Ashkenazi Jewish	European	na	
Repetitive bases	SINE	4,385,917	2,625,350	1.7	1.7
	Retroposon	18,500	18,506	1.0	1.0
	LINE	6,456,888	6,378,323	1.0	1.0
	LTR	4,613,537	4,604,368	1.0	1.0
	DNA/Rolling-circle	4,387,030	2,626,425	1.7	1.7
	Satellite	14,522,636	1,578,773	9.2	9.2
	Simple/Low-complexity	21,568,381	1,124,311	19.2	19.2
	Other	972,612	705,062	1.4	1.4
	All repeat classes	53,004,524	17,501,283	3.0	3.0
	% repetitive	84.86%	30.58%	2.8	2.8

Table 5.2: Table: GRCh38 Comparison. Excl = Exclusively found in each assembly.

Chapter 6

Gene annotation on a human pangenome

Background

The reference human genome, currently GRCh38 (released in December 2013), forms an incredibly important backbone to human genomics research. However, GRCh38 is not a complete reference. It contains many gaps (995 gaps containing 115 Mb of sequence, approximately 8 percent of the genome) of sequence that is missing entirely, and it also contains some errors, at both a base-pair level as well as at structural levels [122]. It also does not represent a true human genome, as it is a mosaic of multiple individuals.

Clearly, it is time for some updates to the human reference genome. Fortunately, DNA sequencing and assembly technologies have advanced to the point where it

is possible to make much more contiguous assemblies, covering more of the genome with fewer gaps. The assembly of the first fully complete, telomere-to-telomere genome for the CHM13 cell line, described in the previous chapter, is the first attempt at creating an updated human reference genome. There are no longer any gaps in the assembly, and portions of the genome that were entirely absent before like the centromeres and ribosomal DNA arrays are now represented. However, the CHM13 genome has some limitations as well, mainly that it was derived from a hydatidiform mole cell line, which resulted in a nearly entirely homozygous genome and therefore does not represent a true genome that would be found in a diploid human. Regardless, the T2T-CHM13 assembly already shows improvements in genomic analyses, for example discovering 3.7 million additional single-nucleotide polymorphisms (SNPs) in regions non-syntenic to GRCh38 and better representing the true copy-number variants (CNVs) of 1000 Genomes Project (1KG) samples when compared to GRCh38 (1000 Genomes Project Consortium et al., 2015; Aganezov et al., 2022).

Multiple diploid genome assemblies will be needed to address the remaining limitations of the current reference genomes, GRCh38 and CHM13. A single genome assembly will never be able to represent the spectrum of genetic diversity across the human species. This can lead to biases in analysis, generally termed "reference bias". This is particularly harmful when analyzing genomic data from individuals that are from underrepresented populations, as their genomes are more likely to differ from the reference. As a solution, we could have many human genome "references", chosen to include a large amount of genetic diversity. These many genomes could be aligned

together to create a *pangenome*. The input assemblies should be of the highest quality possible, so that the resulting pangenome is also of the highest quality.

The Human Pangenome Reference Consortium

The Human Pangenome Reference Consortium (HPRC) was formed with the goal of creating high-quality diploid genome assemblies from a diverse set of individuals in order to build a pangenome reference. The HPRC aims to eventually sequence and assemble 350 diverse individuals (700 haplotypes). The following chapter describes an overview of the work completed in the first few years of the HPRC's existence, and details my contributions to the consortium thus far.

HG002 Assemblathon

The first step to creating a human pangenome is assembling the genomes that will comprise it. In order to determine the best pipeline for doing so, the HPRC organized a bake-off where many teams from different institutions assembled the HG002 diploid genome, using whichever methods they wanted. The results of this "assemblathon" would provide the recipe for generating sequencing information and producing automated assemblies for the initial subset of individuals from the HPRC panel.

There were many different sequencing data types available. These included long-read sequencing for the construction of the initial assembly contigs (Oxford Nanopore long reads, PacBio HiFi reads), and reads that provided long-range link information (10X linked reads, Hi-C linked reads, BioNano optical maps, Strand-seq). High cov-

erages for all of the sequencing types were generated to test the effects of different coverage levels; and participants were also meant to test coverages downsampled to manufacturer recommendations in order to aid in comparison across the produced assemblies. Parental samples, from the father (HG003) and the mother (HG004), were included as well.

Contributions

I participated on the UC Santa Cruz team, where we chose to use the Shasta pipeline (as described in a previous chapter) to assemble the HG002 genome. I specifically contributed by running parts of the Shasta pipeline, and compared different methods of scaffolding with HiC proximity ligation reads (HiRise, Salsa2) on the resulting Shasta draft assemblies. I also ran some of the QC for the assemblies, including running QUAST to get broad assembly QC results and running CAT to get gene annotation results. I was a coauthor on the paper in which this research was recently published, "Semi-automated assembly of high-quality diploid human reference genomes" which came out in Nature in October 2022 [126].

Results

There were a total of 23 assemblies produced from 14 different groups. There were 12 assembly algorithms used: Canu and HiCanu, CrossStitch, DipAsm, FALCON Unzip, Flye, hifiasm, MaSuRCA, NECAT, Peregrine, Shasta, and wtdbg2. A summary of the continuity, phasing, and base level accuracy for each of the assemblies can be

found in Figure 6.1.

There were some key takeaways from the comparison of all approaches. The best approaches used the highly accurate PacBio HiFi reads. They also took advantage of the parent-child data to do graph-based haplotype phasing. The best diploid HG002 assembly was very high-quality. It was very contiguous, with approximately four gaps per chromosome. The best pipeline was determined to be using hifiasm (with trios) to generate the contigs for the first 47 individuals in the HPRC panel.

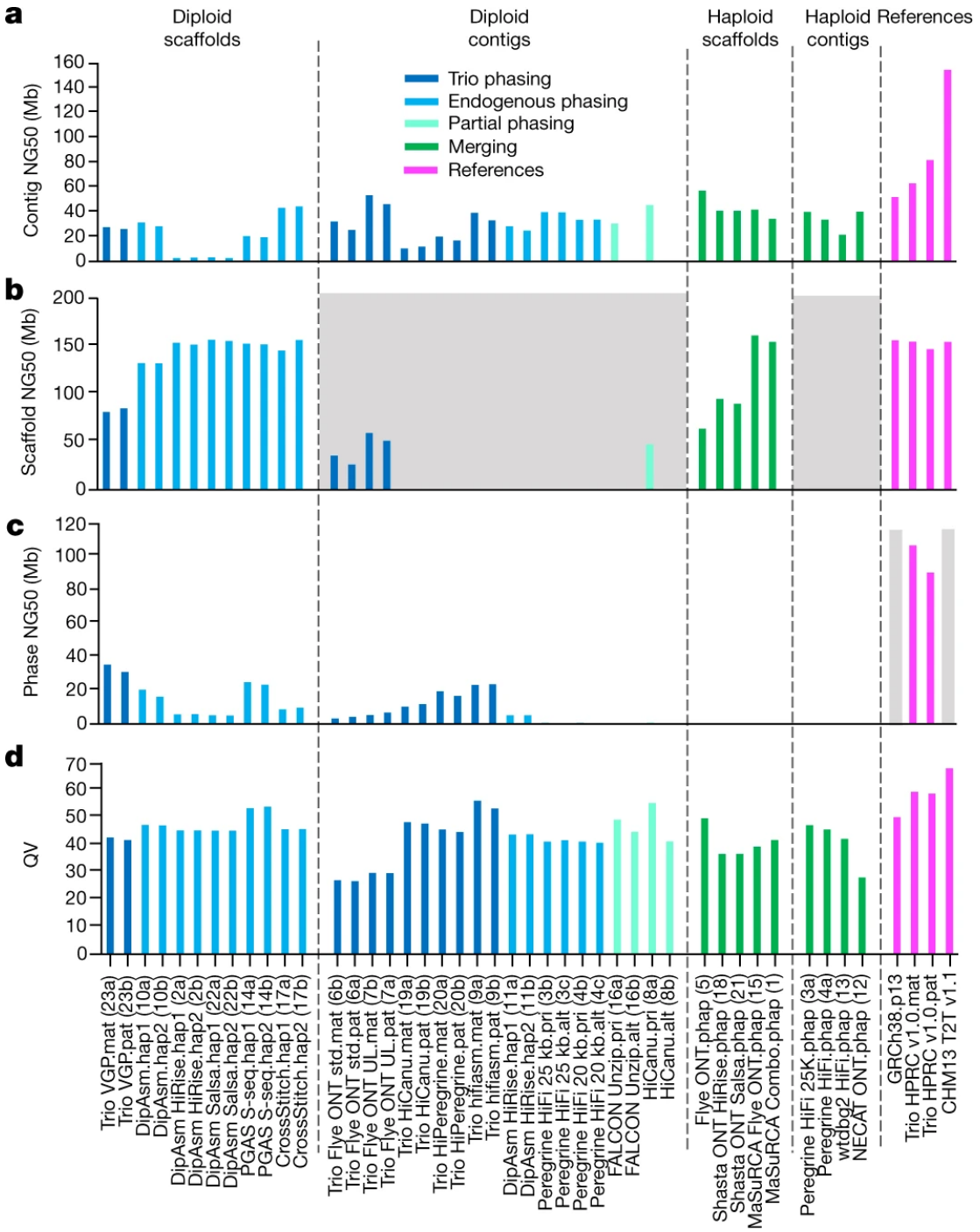

Figure 6.1: **Assembly continuity, phasing and base call accuracy metrics.** a, Contig NG50 values. b, Scaffold NG50 values. c, Haplotype phase block NG50 values. d, QV base call accuracy; as an example, QV60 is about one error per megabase. The dashed lines separate the assemblies into the four major categories as described in Table 1. The colours designate the type of haplotype phasing performed: Trio phasing using parental data, endogenous phasing using self-data, partial endogenous phasing, merging of haplotypes, and final references with various phasing approaches. The grey shaded regions in b are not applicable for scaffold metrics, as these are contig-only assemblies; however, the Flye assembler inserts gaps into contigs where there is uncertainty of a repeat sequence, and the *purge$_d$ups* function applied to the HiCanu contigs removes false duplications within contigs and creates a gap in the removed location. The grey shading in c indicates not applicable for phase blocks, because GRCh38 has many haplotypes and CHM13 is from a haploid (hap) cell line. The numbers in parentheses along the x axis are the assembly numbers. alt, alternate; mat, maternal; pat, paternal; phap, psuedo-haplotype; pri, primary; std., standard ONT read length; S-seq., Strand-Seq; UL., ultra-long ONT read length.

A draft pangenome using the HPRC Year 1 assemblies

Goals for the following year of the HPRC

The next phase of the HPRC project was to apply the assembly pipeline to a panel of 47 individuals, selected to represent global genetic diversity. Subsequently, these phased assemblies would be used to construct a draft human pangenome (also included were two haploid, reference-quality assemblies: GRCh38 and T2T-CHM13).

Contributions

My main contribution to the HPRC was performing gene annotation on the pangenome and using it to compare how genes differ can between individuals. I ran CAT [98] on the resulting pangenome graphs to lift over the reference GENCODE v38 annotations onto each of the individual haplotype assemblies presented in the graph. I analyzed the quality of these annotations, focusing on things such as the numbers genes and transcripts for each biotype that were mapped, and the quality of these mappings (identifying potential errors such as frameshifts and early stops within the transcripts). I compared these sets of annotations to the annotation sets provided by Ensembl, which used a different method of creating them. I also worked on analysis of the copy number variation present within the panel of genomes, alongside Mark Chaisson. Finally, I worked on the creation of the UCSC Genome Browser resources associated with this work, alongside Julian Lucas.

This work will published shortly. Currently, it is available as a preprint called

"A Draft Human Pangenome Reference" [127] and is in the second round of review at Nature. For this paper, I performed the work described above. I also contributed to writing the relevant sections of the main text, and I made the tables and figures associated with those sections.

Results

The following sections describe some of the published results. I give a brief description of the general results, but focus mainly on the results for my contributions regarding gene annotation on the pangenome.

Assembly results

The HPRC created 47 fully phased diploid assemblies for the genomes that had been selected for the panel. These genomes and their cell lines all met the criteria that they were karyotypically normal, with low passage, and had sequencing data for the parents available. Samples were then prioritized to maximize genetic diversity. Each sample was deeply sequenced with the following methodologies: PacBio HiFi reads, ONT long reads, Bionano optical maps, and Hi-C proximity ligation sequencing. Previously generated high-coverage Illumina short read data sets were also used.

The assemblies were made using Trio-Hifiasm [63], which uses PacBio HiFi reads and Illumina short reads from the parents to create assemblies that are nearly fully phased. An automated assembly quality control pipeline was developed and run on each of the assemblies. The average NG50 of the assemblies was 40 Mb, and the

average quality value (QV) assessed using Yak and parental kmers from Illumina reads was 53.57. To evaluate phasing, the average switch error rate was 0.67% and the average Hamming error rate was 0.79%. The Flagger pipeline was developed to assess reliability and misassemblies using read alignments, and only 0.88% of each assembly was flagged as unreliable.

Pangenome construction

After creating the initial assemblies, a few different methods were used to construct a pangenome containing all of the assemblies. Minigraph starts with an assembly, here GRCh38, and progressively adds additional assemblies, adding SVs larger than 50 bases and complex variants. Minigraph-Cactus (MC) extends this pangenome by using Cactus [109] to add a base-level alignment between the assemblies. Long sequences unalignable to other portions of the graph were clipped out. In contrast, the Pangenome Graph Builder (PGGB) uses an all-to-all alignment of the assemblies to construct the pangenome, not starting from a single reference (though both GRCh38 and CHM13 are included.) Repeat copies are collapsed into a single copy in the graph, due to the difficulty of placing these copies in an all-to-all alignment. It does not remove any clipped regions like the MC graph, so it is larger in size and complexity. We focused on the MC graph for gene annotation, though we tested out the PGGB graph as well

Gene annotation results

I ran CAT (Comparative Annotation Toolkit) [98] on the resulting pangenomes to lift over the GENCODE v38 annotations onto each of the individual haplotype assemblies. Because CAT uses the alignments in the pangenome to project the transcripts from the reference genome (GRCh38) onto each of the targets, the resulting annotations are only on portions of the genome that were alignable between the reference and each target. This means that this method of gene annotation can be viewed as another form of quality assessment on the graph construction (in addition to quality assessment of the assemblies themselves). Though I ran CAT on each of the pangenomes created, only the results for the Minigraph-Cactus graph that used GRCh38 as the starting reference are presented.

In summary, CAT lifted and annotated a median of 99.5% of the 86,757 protein-coding transcripts onto each assembly. The results of this can be seen in the below figures, Figure 6.2 and 6.3. These figures also include the results for the Ensembl annotations and annotations on the PGGB graph. I also analyzed the number of transcripts that included mutations that caused frameshifts and early stop codons, seen in Figure 6.4. In total, there were a median of 72 transcripts per assembly that had frameshifts in the MANE isoform, and were also confirmed by Illumina variant calls, and a median of 31 transcripts per assembly that had nonsense mutations in the MANE isoform (also confirmed by Illumina variant calls). Relevant methods for this are described in more detail in the section 'Gene annotation QC pipeline'.

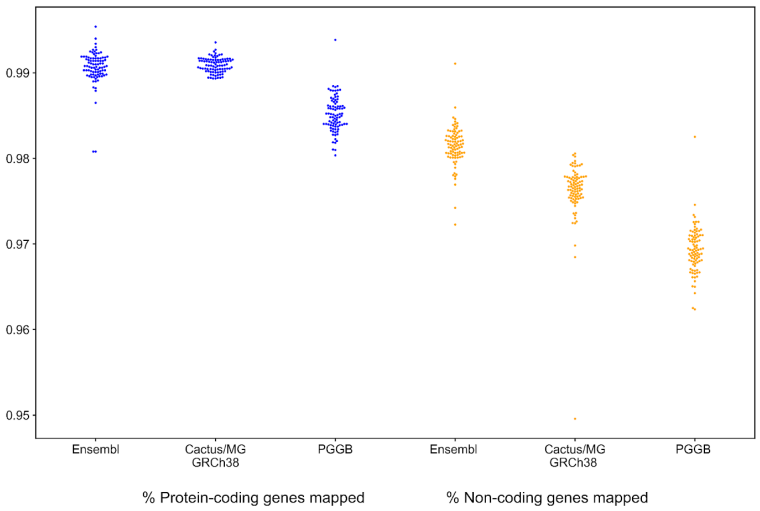

Figure 6.2: **Gene mapping in the pangenome graphs.** The first three show the percentage of protein-coding genes from GENCODE v38 able to be mapped in the gene annotation sets from Ensembl, CAT run on the MC graph based on GRCh38, and CAT run on the PGGB graph. The second three show the percentage of non-coding genes from GENCODE v38 able to be mapped on the same annotation sets.

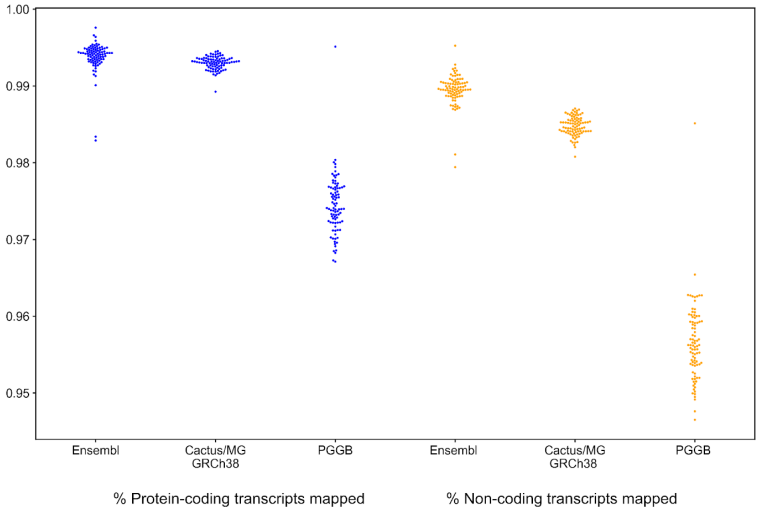

Figure 6.3: **Transcript mapping in the pangenome graphs.** The first three show the percentage of protein-coding transcripts from GENCODE v38 able to be mapped in the gene annotation sets from Ensembl, CAT run on the MC graph based on GRCh38, and CAT run on the PGGB graph. The second three show the percentage of non-coding transcripts from GENCODE v38 able to be mapped on the same annotation sets.

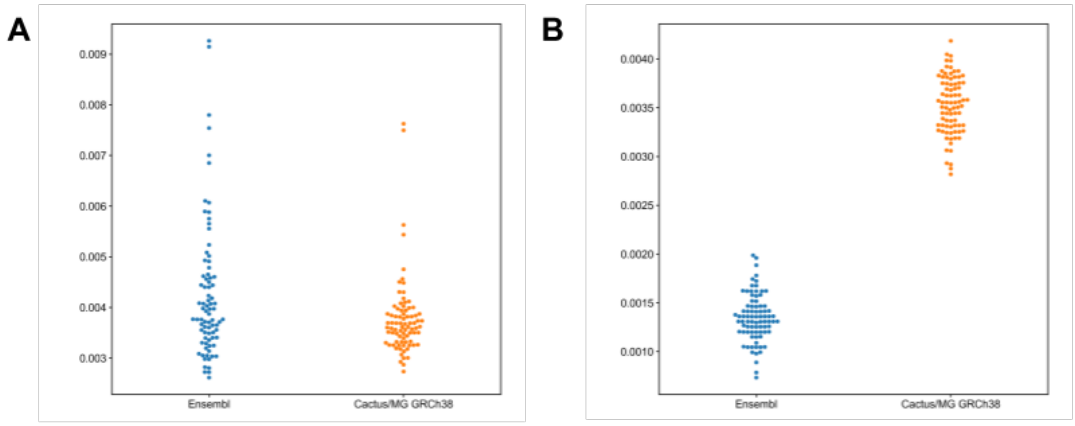

Figure 6.4: **Genes with frameshift mutations or nonsense mutations in the pangenome graphs.** A) Fraction of canonical transcripts with a frameshifting indel from GENCODE v38 in the Ensembl annotations and the CAT annotations of the GRCh38-based pangenome graph.B) Fraction of canonical transcripts with early in-frame stop codons in the Ensembl annotations and the CAT annotations of the GRCh38-based pangenome graph.

Comparison to alternative gene annotation sets

There was another gene annotation pipeline run on the diploid genome assemblies, designed by Ensembl (EBI). The following contains the description of the Ensembl pipeline and the results, taken from the text of the HPRC paper. I performed the analysis regarding the completion and quality stats for their pipeline, and compared Ensembl's results to the results from CAT.

Ensembl pipeline description

We developed a new Ensembl mapping pipeline to annotate GENCODE (Frankish et al., 2021) genes and transcripts within each new haploid assembly (Methods). To create high-confidence annotations, the pipeline clusters and maps spatially proximal genes in parallel (to help avoid issues with individually mapping near identical par-

alogues) and attempts to resolve inconsistent mappings by both considering the synteny of the gene neighborhood in relation to the GRCh38 annotation and the identity and coverage of the underlying mappings. A median of 99.07% of protein-coding genes (minimum of 98.08%, maximum of 99.40%) and 99.42% of protein-coding transcripts (minimum of 98.29%, maximum of 99.66%) were unambiguously identifiable in each of the HPRC assemblies (Figure 6.5; Supplementary Table Ensembl Annotation). Similarly, a median of 98.16% of non-coding genes (minimum of 97.23%, maximum of 98.60%) and 98.96% of non-coding transcripts (minimum of 97.94%, maximum of 99.28%) were similarly annotated.

By way of comparison, running this pipeline on T2T-CHM13 gives comparable, if slightly higher, results: we annotated 99.54% and 99.76% of protein-coding genes and transcripts, and 99.11% and 99.52% of non-coding genes and transcripts in T2T-CHM13. Intersecting the HPRC/HPRC+ annotations with the assembly reliability predictions, a median of 99.53% of gene and 99.79% of transcript annotations occurred wholly within reliable regions, indicating that the vast majority of the annotated haplotypes are structurally correct.

To examine transcriptome base accuracy, we looked for nonsense and frameshift mutations in the set of canonical transcripts (one representative transcript per gene; Methods and Supplementary Table Ensembl Annotation, Figure 6.4). We found a median of 25 nonsense mutations per assembly, supporting the idea that there is a low level of base-level substitution error. A median of 21 (84%) of these nonsense mutations per assembly are supported by the independently generated Illumina variant

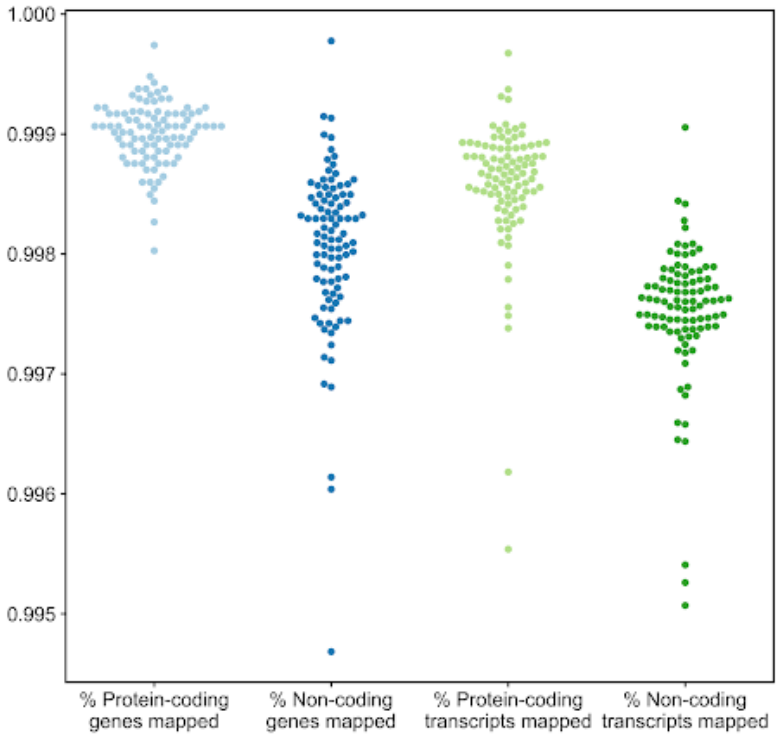

Figure 6.5: **Ensembl mapping pipeline results**. Percentages of protein-coding and non-coding genes and transcripts annotated from the reference set in each of the HPRC assemblies. Orange points represent T2T-CHM13 for comparison.

call sets. We found a median of 72 frameshifts (0.37% of transcripts) mutations per genome, with a median of 67 of these being high-confidence frameshifts not occurring in the leading 5' or 3' ends of the transcript. A median of 58 (80%) of these frameshifts per assembly are supported by the same Illumina call sets. These numbers are within the range of previously reported numbers of loss-of-function mutations (between 10-150 per person, depending on the level of conservation of the mutation) (1000 Genomes Project Consortium et al., 2015; MacArthur et al., 2012). Some of the non-confirmed frameshifts

and nonsense mutations (a median of 14 frameshifts and 4 nonsense mutations per assembly, or one error per ˜1.7 million reference transcriptome bases) are likely assembly errors.

Gene annotation QC pipeline

QC of a single gene set in comparison to a reference

Frameshifts

For the Ensembl and CAT gene annotation sets, we identified the locations of frameshifting insertions and deletions by iterating over the coding sequence of each transcript and looking for any gaps in the alignment. If the gap had a length that was not a multiple of 3, and its length was less than 30 base pairs long (to remove likely introns from consideration), the gap is determined to be a frameshift and its location is saved to a BED file.

Nonsense mutations

We also analyzed the number of nonsense mutations that would cause early stop codons in both the Ensembl and CAT gene annotation sets. We identified the nonsense mutations by iterating through each codon in the coding sequence of the predicted transcripts, and if there was an early stop codon before the canonical stop codon at the end of the transcript, we saved the location in a BED file.

Though there may be observed errors in frameshifting and nonsense mutations

in the output gene annotations, consideration should be taken to determine their source. There are a few possibilities: errors arising from the gene annotation, indel errors present within the assemblies themselves that would then lead to downstream indel errors, and true observed frameshifts/nonsense mutations that do lead to true frameshifted or truncated proteins.

Validation of mutations with Illumina

For both sets of mutations, we then lifted over the coordinates of the mutations to be on the GRCh38 reference so that we could use existing variant callsets on GRCh38. We used halLiftover to lift over each set of coordinates, using the GRCh38-based HAL file from the cactus-minigraph alignment. Then, we used bedtools intersect to intersect with the variant call file for each of the assemblies.

Sample commands:

```
halLiftover GRCh38-f1g-90-mc-aug11.hal <GENOME_NAME> <MUTATION_BED_FILE>
GRCh38 <LIFTED_OVER_BED_FILE>
bedtools intersect -wo -a <LIFTED_OVER_BED_FILE> -b <SAMPLE_MERGED_VCF> >
<OVERLAP_OUTPUT_TXT_FILE>
```

The VCF files used in this intersection were downloaded from the 1KG:

```
ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/
1000G_2504_high_coverage/working/20201028_3202_raw_GT_with_annot/
20201028_CCDG_14151_B01_GRM_WGS_2020-08-05_chr\$i.recalibrated_variants.vcf.gz
```

where $i was replaced with each chromosome number. From there, each chromosome vcf was split so that each sample was in its own file using bcftools view. The chromosome files for each sample were combined into one VCF using bcftools concat.

QC to compare two non-reference gene sets

I also compared the Ensembl and CAT gene sets directly to each other, in order to determine the number of genes and transcripts that were annotated by both pipelines, and the numbers that were annotated by only one of the pipelines. A median of 10 protein-coding genes (51 transcripts) per assembly were annotated only by CAT, compared to a median of 156 protein-coding genes (521 transcripts) annotated only by Ensembl. The vast majority (99.7%) of genes were successfully annotated by both pipelines. I also determined the number of genes and transcripts that were annotated at different loci between the two pipelines, which resulted in 179 protein-coding genes (360 transcripts).

The script used to perform this analysis can be found at `https://github.com/mhaukness-ucsc/gene_annotation_compare`.

Discovering duplicated gene copies

I collaborated with Mark Chaisson to identify dupicated gene copies in the pangenome. I ran Liftoff [125] using extra copy mode to identify additional copies of genes. Mark supplemented this set with additional copies from his protocol. The methods for both protocols are described in more detail in the next section. Because

paralogs of genes in the same gene family can have very similar protein sequences, we assigned each gene under consideration to a "gene family", where genes had high similarity with each other. We did all further analysis with respect to those gene families. The next section is an excerpt from the relevant section of the paper describing the results of this analysis. I included larger versions of two of the plots included in one of the paper figures, so they appear twice. The section after describes the methods used for this analysis, and is also an excerpt from the paper.

Gene family results

There are 1,529 protein-coding gene families within the Flagger predicted reliable regions of the full set of assemblies that have a gain in copy number in at least one genome (Figure 6.8B). Each assembly has an average of 44 genes with a gain in copy number relative to GRCh38 within its predicted reliable regions, with a bias towards rare, low-copy CNVs (Figure 6.8C); 80% of CNV genes appear in a single haplotype. Previous studies using read depth found that rare CNVs occur generally outside of regions annotated as being enriched in SDs (Sudmant et al., 2010). The genome assemblies confirm this observation in sequence-resolved CNVs. When stratifying duplicated genes based on allele frequency (AF) into singleton (present in one haplotype), low frequency ($< 10\%$), and high frequency, 13% (159/1,181) of the singleton CNVs map to SDs as annotated in GRCh38. Duplicated genes with a higher population frequency have a greater fraction in SDs: 40% (86/214) of low-frequency, and 81% (148/184) of high frequency. 63 genes are CNVs in 10% or more of haploid assemblies, and 17 genes are

amplified in the majority of individuals relative to GRCh38 (Figure 6.6; Supplementary Table AdditionalCopyNumberVariants).

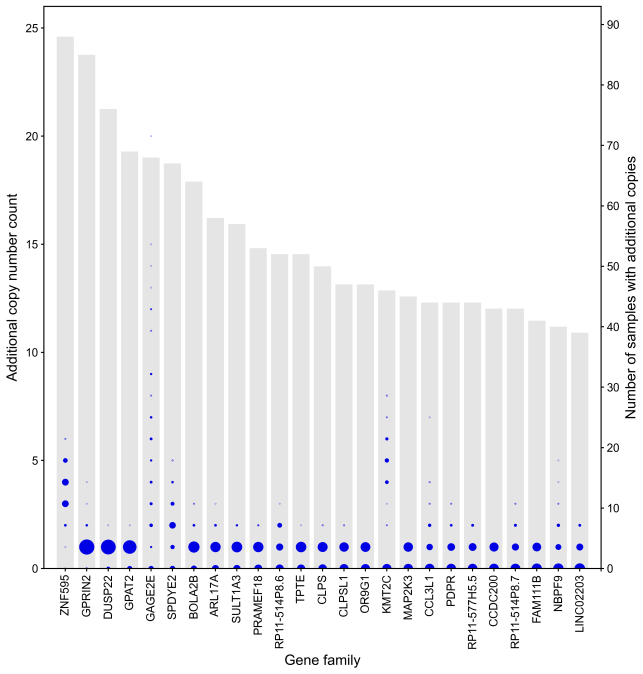

Figure 6.6: **The top 25 most commonly CNV genes or gene-families in the HPRC/HPRC+ assemblies, ordered by the number of samples with additional copies relative to GRCh38.** Grey bars represent the number of samples with additional copies. Blue circles represent the number of additional copies per sample, with the size of the circle proportional to the number of samples.

Many of these genes are individually highly copy-number polymorphic and part of complex tandem duplications (Figure 6.7). For example, the gene GPRIN2 is known to be copy-number polymorphic (Handsaker et al., 2015) based on read depth, and has

sequence resolution of 1-3 additional copies duplicated in tandem in the pangenome (Figure 6.8F). The gene SPDYE2 is similarly resolved as 1-4 additional copies duplicated in tandem (Figure 6.8G). Other copy number variable genes are not contiguously resolved and reflect limitations of the current assemblies (see Porubsky et al. companion). For example, the defensin gene DEFB107A has 3-8 additional copies assembled across all samples, however this gene is assembled into 3-7 separate contigs that do not reflect the global organization of this gene.

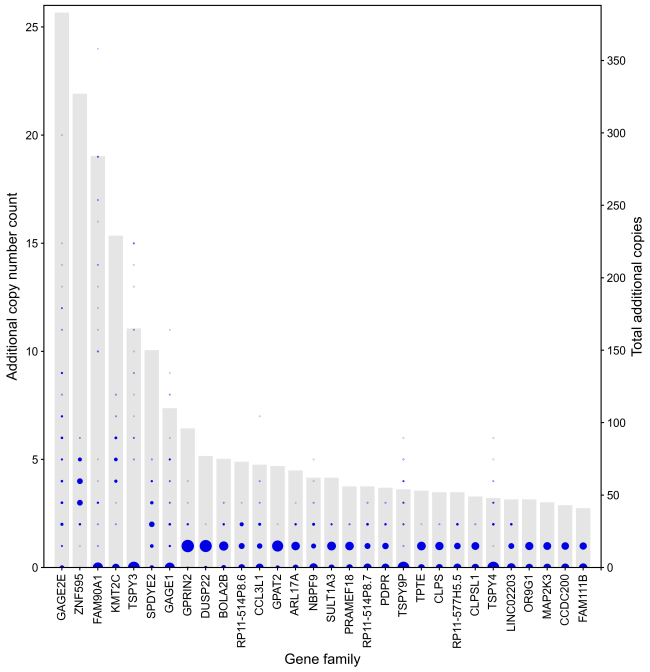

Figure 6.7: **The top 30 most individually CNV genes or gene families in the HPRC/HPRC+ assemblies, ordered by total number of additional copies observed.** Blue circles again represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. The Grey bars represent the total number of additional copies summed over the samples.

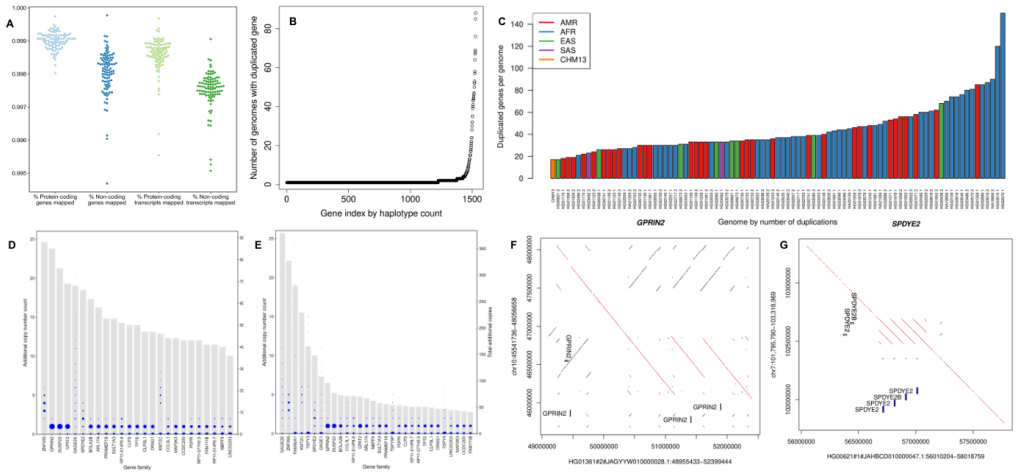

Figure 6.8: **Transcriptome annotation of the assemblies.** A) Ensembl mapping pipeline results. Percentages of protein-coding and non-coding genes and transcripts annotated from the reference set in each of the HPRC assemblies. Orange points represent T2T-CHM13 for comparison. B) Assembled gene duplications per genome. The number of genomes containing a duplicated gene for 1529 protein-coding gene duplications indexed by increasing copy number, observed in the predicted reliable regions of the HPRC/HPRC+ genomes. C) Number of distinct duplicated genes or gene families per phased assembly relative to the number of duplicated genes annotated in GRCh38 (152). The GRCh38 gene duplications reflect families of duplicated genes, while the counts in other genomes reflect gene duplication polymorphisms. The assemblies are color coded according to their population of origin. D) The top 25 most commonly CNV genes or gene-families in the HPRC/HPRC+ assemblies, ordered by the number of samples with additional copies relative to GRCh38. Grey bars represent the number of samples with additional copies. Blue circles represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. E) The top 30 most individually CNV genes or gene families in the HPRC/HPRC+ assemblies, ordered by total number of additional copies observed. Blue circles again represent the number of additional copies per sample, with the size of the circle proportional to the number of samples. The Grey bars represent the total number of additional copies summed over the samples. F) Dotplot illustrating haplotype-resolved GPRIN2 gains in the HG01361 assembly relative to GRCh38. G) Dotplot illustrating SPDYE2/SPDYE2B haplotype resolved gains within a tandem duplication cluster of the HG00621 assembly relative to GRCh38.

Gene Duplication Analysis Methods

Duplicated genes were detected as multi-mapped coding sequences using Liftoff (Shumate & Salzberg, 2020) supplemented by a complementary approach (gb-map) using multi-mapped gene bodies. The combined set was formed by including all liftoff gene duplications and duplicated genes detected by gb-map. Liftoff We ran Liftoff (`commit 35a4e5536414c4ac3b49873f427388d54bc24fd7`) to annotate extra gene copies in each of the assemblies. Liftoff was run with the flag `-sc=0.90` to find additional copies of genes, with an identity threshold of at least 90%. An example command is below:

```
liftoff -p 10 -sc 0.90 -copies -db <GENCODE_V38_DATABASE>
    -u <UNMAPPED_FILE> -o <OUTPUT_GFF3> -polish <GENOME_FASTA> <GRCh38_FASTA>
```

The additional copies of the genes are identified as such in the output gff3 with the field `extra_copy_number` (equal to anything other than 0). For this analysis, we also only considered genes that were multi-exon, protein-coding genes. The additional gene copies were further filtered to remove any genes outside of the "reliable", haploid regions as determined by the Flagger pipeline.

gb-map

The gene-body mapping pipeline identifies duplicated genes by first aligning transcripts of protein coding and pseudogenes (GENCODE v38) to each assembly, and then multi-mapping the genomic sequences of each corresponding gene. Alignments of at least 90% identity and 90% of the length of the original duplication are considered

candidate duplicated genes. Candidates are removed if they overlap previously mapped transcripts from other genes, low-quality duplications, and genes identified through CAT and liftoff analysis. Gene family analysis To account for gene duplications in high-identity gene families, gene families are identified based on sequence alignments from gb-map. Genes that map reciprocally with 90% identity and 90% length are considered a gene family. A single gene is selected as the representative gene for the family, and any gene duplication in the family is counted towards that gene.

Future Directions

The work on the gene family copy number analysis will hopefully be extended in a future paper. We would like to do further analysis into these genes, particularly looking at any coding differences within the copies of these genes.

Building UCSC Genome Browser resources

The HPRC pangenome and all its associated resources need to be publicly available for other users to explore. The UCSC Genome Browser is a tool to graphically visualize genomes and associated tracks. These tracks can cover a broad range of data; some examples relevant to this paper include gene annotations, SNPs, and genome alignments, among other things.

As a part of the HPRC project, I built the basis for the Assembly Hub which can be used in the UCSC Genome Browser to visualize each of these assemblies and many results from downstream analyses. I added all versions of the gene annotation

tracks (both the ones from CAT and Ensembl) and the alignments between each genome in the different pangenome versions. Julian Lucas has also put in a great amount of effort on this project, leveraging the files in my hub, which serves as a 'development' hub, to create a public-facing hub that includes additional tracks such as segmental duplications, tandem repeats, and the reliable regions determined by Flagger. This public hub can be accessed at the following URL: `http://hprc-browser.ucsc.edu/`

Bibliography

[1] Jeffrey Rogers and Richard A. Gibbs. Comparative primate genomics: emerging patterns of genome content and dynamics. *Nature Reviews Genetics*, 15(5):347–359, Aug 2014.

[2] Jana Ebler, Marina Haukness, Trevor Pesout, Tobias Marschall, and Benedict Paten. Haplotype-aware diplotyping from noisy long reads. *Genome Biology*, 20(1):116, 2019.

[3] Justin M Zook, Jennifer McDaniel, Nathan D Olson, Justin Wagner, Hemang Parikh, Haynes Heaton, Sean A Irvine, Len Trigg, Rebecca Truty, Cory Y McLean, et al. An open resource for accurately benchmarking small variant and reference calls. *Nature Biotechnology*, 37(5):561, 2019.

[4] Spyros Oikonomopoulos et al. Benchmarking of the Oxford Nanopore MinION sequencing for quantitative and qualitative assessment of cDNA populations. *Scientific Reports*, 6:31602, Aug 2016.

[5] Hengyun Lu, Francesca Giordano, and Zemin Ning. Oxford Nanopore MinION Sequencing and Genome Assembly. *Genomics, proteomics & bioinformatics*, 14(5):265–279, Oct 2016.

[6] F. Sanger, S. Nicklen, and A. R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

[7] Michael L. Metzker. Sequencing technologies — the next generation. *Nature Reviews Genetics*, 11(1):31–46, 2009.

[8] Job Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing chromosome conformation. *Science*, 295(5558):1306–1311, 2002.

[9] Harmen J.G. van de Werken, Paula J.P. de Vree, Erik Splinter, Sjoerd J.B. Holwerda, Petra Klous, Elzo de Wit, and Wouter de Laat. 4c technology: Protocols and data analysis. *Methods in Enzymology*, page 89–112, Aug 2012.

[10] Grace X Y Zheng, Billy T Lau, Michael Schnall-Levin, Mirna Jarosz, John M Bell, Christopher M Hindson, Sofia Kyriazopoulou-Panagiotopoulou, Donald A Masquelier, Landon Merrill, and Jessica M et al. Terry. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. *Nature Biotechnology*, 34(3):303–311, 2016.

[11] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature Biotechnology*, 34(5):518–524, May 2016.

[12] Teri A. Manolio, Francis S. Collins, Nancy J. Cox, et al. Finding the missing heritability of complex diseases. *Nature*, 461(7265):747–753, oct 2009.

[13] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A DePristo. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–303, Sep 2010.

[14] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv*, Jul 2012.

[15] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, Aug 2009.

[16] Ryan Poplin, Pi-Chuan Chang, David Alexander, et al. A universal snp and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36(10):983–987, 2018.

[17] Ruibang Luo, Fritz J. Sedlazeck, Tak-Wah Lam, and Michael C. Schatz. A multi-task convolutional deep neural network for variant calling in single molecule sequencing. *Nature Communications*, 10(1), Jan 2019.

[18] Tamas Szalay and Jene A Golovchenko. De novo sequencing and variant calling with nanopores using PoreSeq. *Nature Biotechnology*, 33(10):1087–1091, Oct 2015.

[19] Miten Jain, Ian T Fiddes, Karen H Miga, Hugh E Olsen, Benedict Paten, and Mark Akeson. Improved data analysis for the MinION nanopore sequencer. *Nature Methods*, 12(4):351–356, Apr 2015.

[20] Fei Guo, Dan Wang, and Lusheng Wang. Progressive approach for snp calling and haplotype assembly using single molecular sequencing data. *Bioinformatics*, 34(12):2012–2018, 2018.

[21] Sharon R. Browning and Brian L. Browning. Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics*, 12(10):703–714, oct 2011.

[22] Marcel Martin, Murray Patterson, Shilpa Garg, Sarah Fischer, Nadia Pisanti, Gunnar W Klau, Alexander Schoenhuth, and Tobias Marschall. Whatshap: fast and accurate read-based phasing. *bioRxiv*, page 085050, 2016.

[23] Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E. Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, et al. Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nature Biotechnology*, May 2020.

[24] Kishwar Shafin, Trevor Pesout, Pi-Chuan Chang, Maria Nattestad, Alexey Kolesnikov, Sidharth Goel, Gunjan Baid, Mikhail Kolmogorov, Jordan M. Eizenga, Karen H. Miga, and et al. Haplotype-aware variant calling with pepper-margin-deepvariant enables high accuracy in nanopore long-reads. *Nature Methods*, 18(11):1322–1332, 2021.

[25] John E. Gorzynski, Sneha D. Goenka, Kishwar Shafin, Tanner D. Jensen, Dianna G. Fisk, Megan E. Grove, Elizabeth Spiteri, Trevor Pesout, Jean Monlong, Gunjan Baid, Jonathan A. Bernstein, Scott Ceresnak, Pi-Chuan Chang, Jeffrey W. Christle, Henry Chubb, Karen P. Dalton, Kyla Dunn, Daniel R. Garalde, Joseph Guillory, Joshua W. Knowles, Alexey Kolesnikov, Michael Ma, Tia Moscarello, Maria Nattestad, Marco Perez, Maura R.Z. Ruzhnikov, Mehrzad Samadi, Ankit Setia, Chris Wright, Courtney J. Wusthoff, Katherine Xiong, Tong Zhu, Miten Jain, Fritz J. Sedlazeck, Andrew Carroll, Benedict Paten, and Euan A. Ashley. Ultrarapid nanopore genome sequencing in a critical care setting. *New England Journal of Medicine*, 386(7):700–702, 2022. PMID: 35020984.

[26] Sneha D. Goenka, John E. Gorzynski, Kishwar Shafin, Dianna G. Fisk, Trevor Pesout, Tanner D. Jensen, Jean Monlong, Pi-Chuan Chang, Gunjan Baid, Jonathan A. Bernstein, and et al. Accelerated identification of disease-causing variants with ultra-rapid nanopore genome sequencing. *Nature Biotechnology*, 40(7):1035–1041, 2022.

[27] Geraldine A Van der Auwera, Mauricio O Carneiro, Christopher Hartl, et al. From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics*, pages 11–10, 2013.

[28] Adam Auton, Goncalo Abecasis, et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, Sep 2015.

[29] Wentian Li and Jan Freudenberg. Mappability and read length. *Frontiers in Genetics*, 5:381, 2014.

[30] Nicolas Altemose, Karen H. Miga, Mauro Maggioni, and Huntington F. Willard. Genomic Characterization of Large Heterochromatic Gaps in the Human Genome Assembly. *PLoS Computational Biology*, 10(5):e1003628, May 2014.

[31] David Porubsky, Shilpa Garg, Ashley D Sanders, Jan O Korbel, Victor Guryev, Peter M Lansdorp, and Tobias Marschall. Dense and accurate whole-chromosome haplotyping of individual genomes. *Nat. Commun.*, 8(1):1293, November 2017.

[32] Mark J P Chaisson, Ashley D Sanders, Xuefang Zhao, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *bioRxiv*, page 193144, September 2017.

[33] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36(4):338, 2018.

[34] Fritz J Sedlazeck, Hayan Lee, et al. Piercing the dark matter: bioinformatics of long-range sequencing and mapping. *Nature Reviews Genetics*, page 1, 2018.

[35] Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W Klau, and Alexander Schönhuth. WhatsHap: Weighted haplotype assembly for Future-Generation sequencing reads. *J. Comput. Biol.*, 22(6):498–509, June 2015.

[36] Fei Guo, Dan Wang, and Lusheng Wang. Progressive approach for SNP calling and haplotype assembly using single molecular sequencing data. *Bioinformatics*, February 2018.

[37] Justin Zook, Jennifer McDaniel, Hemang Parikh, et al. Reproducible integration of mul-

tiple sequencing datasets to form high-confidence snp, indel, and reference calls for five human genome reference materials. *bioRxiv*, 2018.

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

[39] Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo Van Iersel, Leen Stougie, Gunnar W Klau, and Alexander Schönhuth. Whatshap: weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509, 2015.

[40] Justin M Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E Mason, Noah Alexander, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific data*, 3:160025, 2016.

[41] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[42] John G. Cleary, Ross Braithwaite, Kurt Gaastra, et al. Comparing variant call files for performance benchmarking of next-generation sequencing variant calling pipelines. *bioRxiv*, 2015.

[43] Volodymyr Kuleshov, Dan Xie, Rui Chen, Dmitry Pushkarev, Zhihai Ma, Tim Blauwkamp, Michael Kertesz, and Michael Snyder. Whole-genome haplotyping using long reads and statistical methods. *Nat Biotechnol*, 32(3):261–266, March 2014.

[44] Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. The Complexity of the Single Individual SNP Haplotyping Problem. *Algorithmica*, 49(1):13–36, August 2007.

[45] Harvey J Greenberg, William E Hart, and Giuseppe Lancia. Opportunities for Combinatorial Optimization in Computational Biology. *INFORMS Journal on Computing*, 16(3):211–231, August 2004.

[46] T H Cantor and C R Cantor. Evolution of protein molecules. *Mammalian protein metabolism*, 1:22–123, 1969.

[47] Heng Li, Bob Handsaker, Alec Wysoker, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[48] Sarah O Fischer and Tobias Marschall. Selecting reads for haplotype assembly. *bioRxiv*, page 046771, 2016.

[49] Jayne Y. Hehir-Kwa, Tobias Marschall, et al. A high-quality human reference panel reveals the complexity and distribution of genomic structural variants. *Nature communications*, 7:12989, 2016.

[50] Jana Ebler, Alexander Schönhuth, and Tobias Marschall. Genotyping inversions and tandem duplications. *Bioinformatics*, 33(24):4015–4023, 2017.

[51] International Human Genome Sequencing Consortium. Correction: Initial sequencing and analysis of the human genome. *Nature*, 412(6846):565–566, 2001.

[52] S. Gnerre, I. Maccallum, D. Przybylski, F. J. Ribeiro, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, 2010.

[53] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):10, 2013.

[54] Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–562, 2002.

[55] 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, 2017.

[56] Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods*, 13(12):1050–1054, 2016.

[57] Nicholas H Putnam, Brendan L O'Connell, Jonathan C Stites, Brandon J Rice, Marco Blanchette, Robert Calef, Christopher J Troll, Andrew Fields, Paul D Hartley, Charles W Sugnet, et al. Chromosome-scale shotgun assembly using an in vitro method for long-range linkage. *Genome Research*, 26(3):342–350, 2016.

[58] Jay Ghurye, Arang Rhie, Brian P. Walenz, Anthony Schmitt, Siddarth Selvaraj, Mihai Pop, Adam M. Phillippy, and Sergey Koren. Integrating hi-c links with assembly graphs for chromosome-scale assembly. *PLOS Computational Biology*, Aug 2019.

[59] Adam C. English et al. Mind the gap: Upgrading genomes with pacific biosciences rs long-read sequencing technology. *PLoS ONE*, 7(11), 2012.

[60] Bruce J. Walker et al. Pilon: An integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS ONE*, 9(11), 2014.

[61] Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *BioRxiv*, page 530972, 2019.

[62] Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540, 2019.

[63] Haoyu Cheng, Gregory T. Concepcion, Xiaowen Feng, Haowen Zhang, and Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature Methods*, 18(2):170–175, Feb 2021.

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

[65] Can Alkan, Bradley P Coe, and Evan E Eichler. Genome structural variation discovery and genotyping. *Nature Reviews Genetics*, 12(5):363, 2011.

[66] Shunichi Kosugi, Yukihide Momozawa, Xiaoxi Liu, Chikashi Terao, Michiaki Kubo, and Yoichiro Kamatani. Comprehensive evaluation of structural variation detection algorithms for whole genome sequencing. *Genome biology*, 20(1):117, 2019.

[67] Jon-Matthew Belton, Rachel Patton McCord, Johan Harmen Gibcus, Natalia Naumova, Ye Zhan, and Job Dekker. Hi–c: a comprehensive technique to capture the conformation of genomes. *Methods*, 58(3):268–276, 2012.

[68] Ester Falconer and Peter M Lansdorp. Strand-seq: a unifying tool for studies of chromosome segregation. In *Seminars in cell & developmental biology*, volume 24, pages 643–652. Elsevier, 2013.

[69] Neil I Weisenfeld, Vijay Kumar, Preyas Shah, Deanna M Church, and David B Jaffe. Direct determination of diploid genome sequences. *Genome research*, 27(5):757–767, 2017.

[70] Miten Jain, Ian T Fiddes, Karen H Miga, Hugh E Olsen, Benedict Paten, and Mark Akeson. Improved data analysis for the minion nanopore sequencer. *Nature methods*, 12(4):351, 2015.

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

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

[73] Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt von Haeseler, and Michael C Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nat Methods*, 15(6):461–468, 2018.

[74] Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas, Abraham Morales-Cruz, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13(12):1050, 2016.

[75] Evan E Eichler, Royden A Clark, and Xinwei She. An assessment of the sequence gaps: unfinished business in a finished human genome. *Nature Reviews Genetics*, 5(5):345, 2004.

[76] Ian T Fiddes, Gerrald A Lodewijk, Meghan Mooring, Colleen M Bosworth, Adam D Ewing, Gary L Mantalas, Adam M Novak, Anouk van den Bout, Alex Bishara, Jimi L Rosenkrantz, et al. Human-specific notch2nl genes affect notch signaling and cortical neurogenesis. *Cell*, 173(6):1356–1369, 2018.

[77] Miten Jain, Hugh E Olsen, Daniel J Turner, David Stoddart, Kira V Bulazel, Benedict Paten, David Haussler, Huntington F Willard, Mark Akeson, and Karen H Miga. Linear assembly of a human centromere on the y chromosome. *Nature biotechnology*, 36(4):321, 2018.

[78] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56, 2012.

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

[80] Jue Ruan. SmartDenovo, https://github.com/ruanjue/smartdenovo.

[81] Jason R Miller, Arthur L Delcher, Sergey Koren, Eli Venter, Brian P Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry, and Granger Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.

[82] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[83] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623, 2015.

[84] Ultra-long reads for chm13 genome assembly, https://github.com/nanopore-wgs-consortium/chm13.

[85] Alla Mikheenko, Andrey Prjibelski, Vladislav Saveliev, Dmitry Antipov, and Alexey Gurevich. Versatile genome assembly evaluation with quast-lg. *Bioinformatics*, 34(13):i142–i150, 2018.

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

[87] Peter H Sudmant, Swapan Mallick, Bradley J Nelson, Fereydoun Hormozdiari, Niklas Krumm, John Huddleston, Bradley P Coe, Carl Baker, Susanne Nordenfelt, Michael Bamshad, et al. Global diversity, population stratification, and selection of human copy-number variation. *Science*, 349(6253):aab3761, 2015.

[88] Justin M Zook, Nancy F Hansen, Nathan D Olson, Lesley M Chapman, James C Mullikin, Chunlin Xiao, Stephen Sherry, Sergey Koren, Adam M Phillippy, Paul C Boutros, et al. A robust benchmark for germline structural variant detection. *BioRxiv*, page 664623, 2019.

[89] D. Y. Brandt, V. R. Aguiar, B. D. Bitarello, K. Nunes, J. Goudet, and D. Meyer. Mapping Bias Overestimates Reference Allele Frequencies at the HLA Genes in the 1000 Genomes Project Phase I Data. *G3 (Bethesda)*, 5(5):931–941, Mar 2015.

[90] T. R. Turner, J. D. Hayhurst, D. R. Hayward, W. P. Bultitude, D. J. Barker, J. Robinson, J. A. Madrigal, N. P. Mayor, and S. G. E. Marsh. Single molecule real-time DNA sequencing of HLA genes at ultra-high resolution from 126 International HLA and Immunogenetics Workshop cell lines. *HLA*, 91(2):88–101, 02 2018.

[91] Sergey Koren, Arang Rhie, Brian P Walenz, Alexander T Dilthey, Derek M Bickhart, Sarah B Kingan, Stefan Hiendleder, John L Williams, Timothy PL Smith, and Adam M Phillippy. De novo assembly of haplotype-resolved genomes with trio binning. *Nature biotechnology*, 36(12):1174, 2018.

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

[93] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[94] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017.

[95] Medaka, https://github.com/nanoporetech/medaka.

[96] Pomoxis, https://github.com/nanoporetech/pomoxis.

[97] Bruce J. Walker, Thomas Abeel, Terrance Shea, Margaret Priest, Amr Abouelliel, Sharadha Sakthikumar, Christina A. Cuomo, Qiandong Zeng, Jennifer Wortman, and Sarah K. et al. Young. Pilon: An integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS ONE*, 9(11):e112963, 2014.

[98] Ian T. Fiddes, Joel Armstrong, Mark Diekhans, Stefanie Nachtweide, Zev N. Kronenberg, Jason G. Underwood, David Gordon, Dent Earl, Thomas Keane, and Evan E. et al. Eichler. Comparative annotation toolkit (cat)—simultaneous clade and personal genome annotation. *Genome Research*, 28(7):1029–1038, 2018.

[99] Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. Gencode reference annotation for the human and mouse genomes. *Nucleic acids research*, 47(D1):D766–D773, 2018.

[100] Felipe A. Simão, Robert M. Waterhouse, Panagiotis Ioannidis, Evgenia V. Kriventseva, and Evgeny M. Zdobnov. Busco: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212, 2015.

[101] Mitchell R. Vollger, Glennis A. Logsdon, Peter A. Audano, Arvis Sulovari, David Porubsky, Paul Peluso, Gregory T. Concepcion, Katherine M. Munson, Carl Baker, Ashley D. Sanders, Diana C.J. Spierings, Peter M. Lansdorp, Michael W. Hunkapiller, and Evan E.

Eichler. Improved assembly and variant detection of a haploid human genome using single-molecule, high-fidelity long reads. *bioRxiv*, 2019.

[102] Aaron M Wenger, Paul Peluso, William J Rowell, Pi-Chuan Chang, Richard J Hall, Gregory T Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D Olson, et al. Highly-accurate long-read sequencing improves variant detection and assembly of a human genome. *bioRxiv*, page 519025, 2019.

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

[104] Samuel Levy, Granger Sutton, Pauline C Ng, Lars Feuk, Aaron L Halpern, Brian P Walenz, Nelson Axelrod, Jiaqi Huang, Ewen F Kirkness, Gennady Denisov, et al. The diploid genome sequence of an individual human. *PLoS biology*, 5(10):e254, 2007.

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

[106] B. Paten, D. Earl, N. Nguyen, M. Diekhans, D. Zerbino, and D. Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome Research*, 21(9):1512–1528, 2011.

[107] J. Harrow, A. Frankish, J. M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B. L. Aken, D. Barrell, A. Zadissa, and S. et al. Searle. Gencode: The reference human genome annotation for the encode project. *Genome Research*, 22(9):1760–1774, 2012.

[108] Klaus-Peter Koepfli, Benedict Paten, and Stephen J. O'Brien. The genome 10k project: A way forward. *Annual Review of Animal Biosciences*, 3(1):57–111, 2015.

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

[110] Edward S. Rice and Richard E. Green. New approaches for genome assembly and scaffolding. *Annual Review of Animal Biosciences*, 7(1):17–40, 2019.

[111] Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino, and David Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome Research*, 21(9):1512–1528, Oct 2011.

[112] Zev N. Kronenberg, Ian T. Fiddes, David Gordon, et al. High-resolution comparative analysis of great ape genomes. *Science*, 360(6393), 2018.

[113] Sam Kovaka, Aleksey V. Zimin, Geo M. Pertea, Roham Razaghi, Steven L. Salzberg, and Mihaela Pertea. Transcriptome assembly from long-read rna-seq alignments with stringtie2. *Genome Biology*, 20(1), Dec 2019.

[114] Wesley C. Warren, R. Alan Harris, Marina Haukness, Ian T. Fiddes, Shwetha C. Murali, Jason Fernandes, Philip C. Dishuck, Jessica M. Storer, Muthuswamy Raveendran, LaDeana W. Hillier, and et al. Sequence diversity analyses of an improved rhesus macaque genome enhance its biomedical utility. *Science*, 370(6523), Dec 2020.

[115] Yafei Mao, Claudia R. Catacchio, LaDeana W. Hillier, David Porubsky, Ruiyang Li, Arvis Sulovari, Jason D. Fernandes, Francesco Montinaro, David S. Gordon, Jessica M. Storer, and et al. A high-quality bonobo genome refines the analysis of hominid evolution. *Nature*, 594(7861):77–81, May 2021.

[116] Koen K A Van Rompay. Tackling hiv and aids: contributions by non-human primate models. *Lab Animal*, 46(6):259–270, 2017.

[117] Heinz Feldmann, Friederike Feldmann, and Andrea Marzi. Ebola: Lessons on vaccine development. *Annual Review of Microbiology*, 72(1):423–446, Aug 2018.

[118] Yang Zhou, Jitendra Sharma, Qiong Ke, et al. Atypical behaviour and connectivity in shank3-mutant macaques. *Nature*, 570(7761):326–331, 2019.

[119] Stuart Huntley, Daniel M Baggott, Aaron T Hamilton, Mary Tran-Gyamfi, Shan Yang, Joomyeong Kim, Laurie Gordon, Elbert Branscomb, and Lisa Stubbs. A comprehensive catalog of human krab-associated zinc finger genes: Insights into the evolutionary history of a large family of transcriptional repressors. *Genome Research*, 16(5):669–677, Jan 2006.

[120] J. H. Thomas and S. Schneider. Coevolution of retroelements and tandem zinc finger genes. *Genome Research*, 21(11):1800–1812, 2011.

[121] Eric S. Lander, Lauren M. Linton, Bruce Birren, Chad Nusbaum, Michael C. Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, and et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[122] Valerie A. Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, Paul A. Kitts, Terence D. Murphy, Kim D. Pruitt, Françoise Thibaud-Nissen, Derek Albracht, and et al. Evaluation of grch38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Research*, 27(5):849–864, 2017.

[123] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, Mitchell R. Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, and et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, Mar 2022.

[124] Nicolas Altemose, Glennis A. Logsdon, Andrey V. Bzikadze, Pragya Sidhwani, Sasha A. Langley, Gina V. Caldas, Savannah J. Hoyt, Lev Uralsky, Fedor D. Ryabov, Colin J. Shew, and et al. Complete genomic and epigenetic maps of human centromeres. *Science*, 376(6588), Apr 2022.

[125] Alaina Shumate and Steven L Salzberg. Liftoff: Accurate mapping of gene annotations. *Bioinformatics*, 37(12):1639–1643, 2021.

[126] Erich D. Jarvis, Giulio Formenti, Arang Rhie, Andrea Guarracino, Chentao Yang, Jonathan Wood, Alan Tracey, Francoise Thibaud-Nissen, Mitchell R. Vollger, David Porubsky, and et al. Semi-automated assembly of high-quality diploid human reference genomes. *Nature*, 2022.

[127] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K. Lucas, Jean Monlong, Haley J. Abel, and et al. A draft human pangenome reference. *bioRxiv*, Jul 2022.

[128] Fritz J Sedlazeck, Zachary Lemmon, Sebastian Soyk, William J Salerno, Zachary Lippman, and Michael C Schatz. Svcollector: Optimized sample selection for validating and long-read resequencing of structural variants. *BioRxiv*, page 342386, 2018.

[129] Data release: Highest-quality, most contiguous individual human genome assembly to date.

[130] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004.

[131] Maria Nattestad and Calvin Bao. GitHub - dnanexus/dot: Dot: An interactive dot plot viewer for comparative genomics.

[132] Zev N Kronenberg, Ian T Fiddes, David Gordon, Shwetha Murali, Stuart Cantsilieris, Olivia S Meyerson, Jason G Underwood, Bradley J Nelson, Mark JP Chaisson, Max L Dougherty, et al. High-resolution comparative analysis of great ape genomes. *Science*, 360(6393):eaar6343, 2018.

[133] Mitchell R Vollger, Glennis A Logsdon, Peter A Audano, Arvis Sulovari, David Porubsky, Paul Peluso, Gregory T Concepcion, Katherine M Munson, Carl Baker, Ashley D Sanders, et al. Improved assembly and variant detection of a haploid human genome using single-molecule, high-fidelity long reads. *BioRxiv*, page 635037, 2019.

[134] 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, and et al. A robust benchmark for detection of germline large deletions and insertions. *Nature Biotechnology*, 38(11):1347–1355, 2020.

[135] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[136] Richard J Anderson and Heather Woll. Wait-free parallel algorithms for the union-find problem. In *STOC*, volume 91, pages 370–380. Citeseer, 1991.

[137] Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Detecting superbubbles in assembly graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 338–348. Springer, 2013.

[138] Chen-Shan Chin, David H Alexander, Patrick Marks, Aaron A Klammer, James Drake, Cheryl Heiner, Alicia Clum, Alex Copeland, John Huddleston, Evan E Eichler, et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature methods*, 10(6):563, 2013.

[139] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature methods*, 12(8):733, 2015.

[140] Benedict Paten, Javier Herrero, Kathryn Beal, and Ewan Birney. Sequence progressive alignment, a framework for practical large-scale probabilistic consistency alignment. *Bioinformatics*, 25(3):295–301, 2008.

[141] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[142] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11—seamless operability between c++ 11 and python, 2016.

Appendix A

List of publications

The following is a succinct list of all of my publications from my time in graduate school. Co-first author publications are denoted with an asterisk.

- ***MarginPhase** (co-first): Ebler, J., Haukness, M., Pesout, T. et al. Haplotype-aware diplotyping from noisy long reads. Genome Biol 20, 116 (2019). `https://doi.org/10.1186/s13059-019-1709-0`
- ***Shasta** (co-first): Shafin, K., Pesout, T., Lorig-Roach, R. et al. Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. Nat Biotechnol 38, 1044–1053 (2020). `https://doi.org/10.1038/s41587-020-0503-6`
- **Rhesus macaque**: Warren, W. C., Harris, R. A., Haukness, M., Fiddes, I. T., Murali, S. C., Fernandes, J., et al. Sequence diversity analyses of an improved rhesus macaque genome enhance its biomedical utility. Science 370, 6523 (2020).

https://doi.org/10.1126/science.abc6617

- **Bonobo**: Mao, Y., Catacchio, C.R., Hillier, L.W. et al. A high-quality bonobo genome refines the analysis of hominid evolution. Nature 594, 77–81 (2021). https://doi.org/10.1038/s41586-021-03519-x

- **T2T**: Nurk, S., Koren, S., Rhie, A. et al. The complete sequence of a human genome. Science 376, 6588, (44-53), (2022). https://doi.org/10.1126/science.abj6987

- **T2T epigenetics**: Altemose, N., Logsdon, G. A., Bzikadze, A. V., et al. Complete genomic and epigenetic maps of human centromeres. Science 376, 6588, (2022). https://doi.org/10.1126/science.abl4178

- **Strand-Seq**: Porubsky, D., Ebert, P., Audano, P.A. et al. Fully phased human genome assembly without parental data using single-cell strand sequencing and long reads. Nat Biotechnol 39, 302–308 (2021). https://doi.org/10.1038/s41587-020-0719-5

- **HPRC bakeoff**: Jarvis, E.D., Formenti, G., Rhie, A. et al. Semi-automated assembly of high-quality diploid human reference genomes. Nature (2022). https://doi.org/10.1038/s41586-022-05325-5

- **HPRC marker paper** (preprint, undergoing revision in Nature): Liao, W., Asri, M., Ebler, J., et al. A Draft Human Pangenome Reference. bioRxiv (2022). https://doi.org/10.1101/2022.07.09.499321

- **Gibbon**: Escalona, M., VanCampen, J., Maurer, N. W., Haukness, M., et al. Whole-genome sequence and assembly of the Javan gibbon (Hylobates moloch). Journal of Heredity (2022). https://doi.org/10.1093/jhered/esac043

- **T2T - chrY**: Rhie, Arang, Sergey Nurk, Monika Cechova, Savannah J. Hoyt, Dylan J. Taylor, Nicolas Altemose, Paul W. Hook et al. "The complete sequence of a human Y chromosome." bioRxiv (2022). https://doi.org/10.1101/2022.12.01.518724

Appendix B

MarginPhase appendix

The following includes information from the supplement for the MarginPhase paper.

Comparison Against High Confidence Truthset

In Figure B.1 we provide a comparison against the GIAB high confidence truthset (within high confidence regions) [37]. In the main manuscript, we present the more performant method for each sequencing technology; here we describe the results for the less performant method.

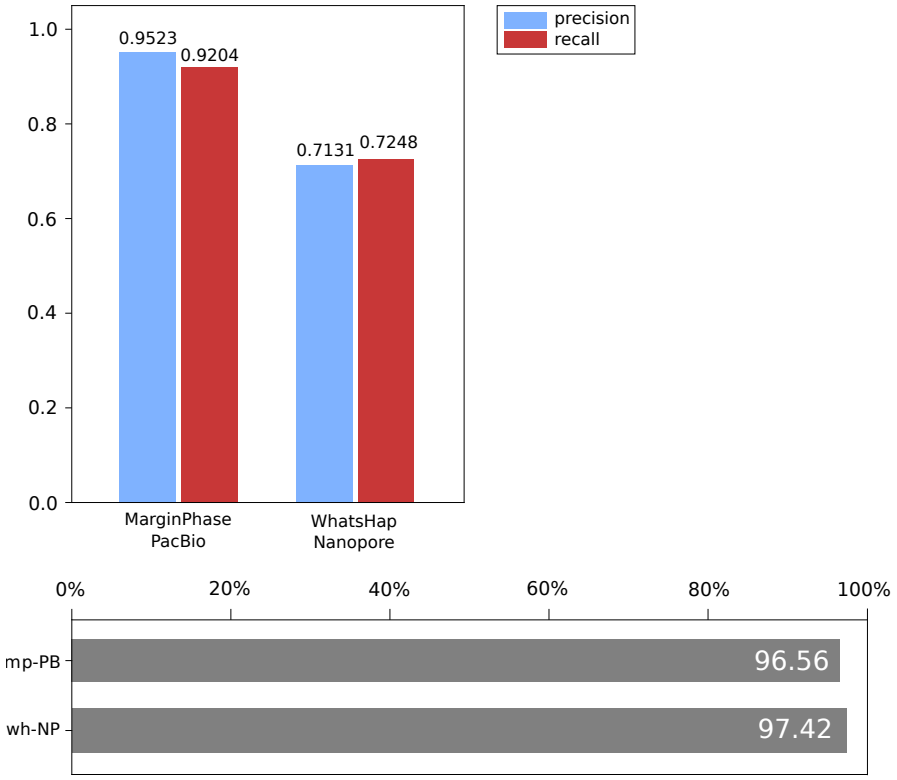

Figure B.1: **Precision and Recall** (Top) of MarginPhase on PacBio and WhatsHap on Nanopore data sets in GIAB high confidence regions. **Genotype Concordance (Bottom)** (wrt. GIAB high confidence calls) of MarginPhase (mp, top) on PacBio and WhatsHap (wh, bottom) on Nanopore.

Cutting and Downsampling Reads

In Figure B.2, we show how the genotyping error behaves as a function of coverage for different lengths of provided read fragments. In the main manuscript, we present the results for the PacBio data, here we give corresponding results for the Nanopore reads. As we observed previously, the genotyping error increases, as the length of the reads decreases due to the lack of information on neighboring variants.

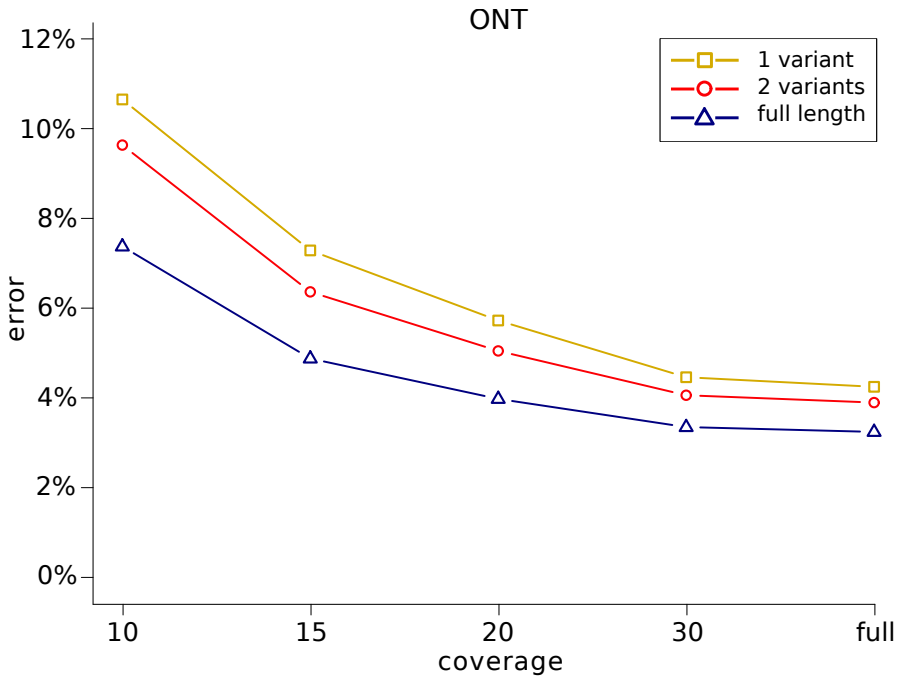

Figure B.2: **Genotyping Errors** (wrt. to GIAB calls) as a function of coverage. The full length reads were used for genotyping (blue) and additionally, reads were cut such as to cover at most two variants (red) and one variant (yellow).

Switch Error Rates (inside of high confidence blocks)

In Table B.1 we describe the switch error rates of our methods for the two sequencing technologies within the GIAB high confidence regions [37]. For the computation of switch errors, we only consider variant positions genotyped as heterozygous in both the callset and ground truth.

chromosome	MP-PB	WH-PB	MP-NP	WH-NP
chr1	0.48%	0.37%	0.32%	0.65%
chr10	0.43%	0.37%	0.27%	0.63%
chr11	0.20%	0.03%	0.07%	0.41%
chr12	0.25%	0.03%	0.09%	0.45%
chr13	0.23%	0.02%	0.07%	0.37%
chr14	0.18%	0.02%	0.06%	0.38%
chr15	0.22%	0.02%	0.09%	0.35%
chr16	0.55%	0.55%	0.45%	1.01%
chr17	0.65%	0.69%	0.54%	1.34%
chr18	0.25%	0.04%	0.09%	0.35%
chr19	0.11%	0.06%	0.21%	1.42%
chr2	0.21%	0.02%	0.08%	0.37%
chr20	0.24%	0.06%	0.15%	0.54%
chr21	0.18%	0.02%	0.07%	0.43%
chr22	0.55%	0.61%	0.45%	1.12%
chr3	0.27%	0.06%	0.11%	0.34%
chr4	0.18%	0.01%	0.06%	0.20%
chr5	0.21%	0.01%	0.09%	0.33%
chr6	0.79%	0.75%	0.53%	0.85%
chr7	0.32%	0.19%	0.19%	0.53%
chr8	0.26%	0.09%	0.10%	0.36%
chr9	0.18%	0.01%	0.07%	0.43%
chrX	0.32%	0.04%	0.12%	0.23%
whole genome	0.32%	0.17%	0.17%	0.50%

Table B.1: Switch error rates of MarginPhase and WhatsHap for each chromosome inside of the GIAB high confidence regions.

Results for Indels

Since WhatsHap and MarginPhase currently cannot detect indels, we re-genotyped the GIAB truth set variants using the WhatsHap implementation (as WhatHap is able to re-genotype given variant positions). We computed the genotype concordance for indels by determining the fraction of correctly genotyped positions among all positions in the truth set for which a genotype could be computed. We also report which fraction of the variants could not be genotyped by our method either due to the position being multi-allelic or because no genotyping information is available at that site after WhatsHaps allele detection and readselection steps. The results are shown in Table B.

	genotype concordance	**not genotyped**
indels (PacBio)	73.82%	6.82%
indels (Nanopore)	55.98%	7.38%

Table B.2: Results from re-typing GIAB truth set indels using WhatsHap on the PacBio and Nanopore reads.

Read Depth Analysis

In Fig B.3 and B.4 we provide an analysis of precision, recall, and f-measure (the harmonic mean of the precision and recall) for our method as a function of read depth. To produce this data, we analyzed our calls with `rtg vcfeval` [42] against the GIAB benchmark small variant calls v3.3.2 [37] in their high confidence region. We annotated the outputted true positive, false positive, and false negative VCF files with the read depth at each variant's reference locus. For each read depth, we counted the number of TP, FP, and FN calls and used them to derive accuracy statistics.

For each sequencing technology and method implementation we plot three pieces of data: in dotted lines, the precision and recall for the calls made at that specific read depth; in solid lines, the precision and recall for all calls made at or above the read depth; and in grey, the amount of calls which were made at each read depth. The vertical line indicates the maximum f-measure considering all variants found at that depth or above. Maximum plotted depth is 100 for PacBio and 75 for ONT; these values were selected as they slightly surpass twice the median depth of the BAMs (46× and 37× respectively).

As is apparent from the plots, the precision and recall are varied at lower depths (less than 20), and at higher depths (roughly 1.5× the median depth), and that these correlate with areas where fewer calls were made. We hypothesize that the decreased accuracies at higher depth are related to copy number variation in the sample.

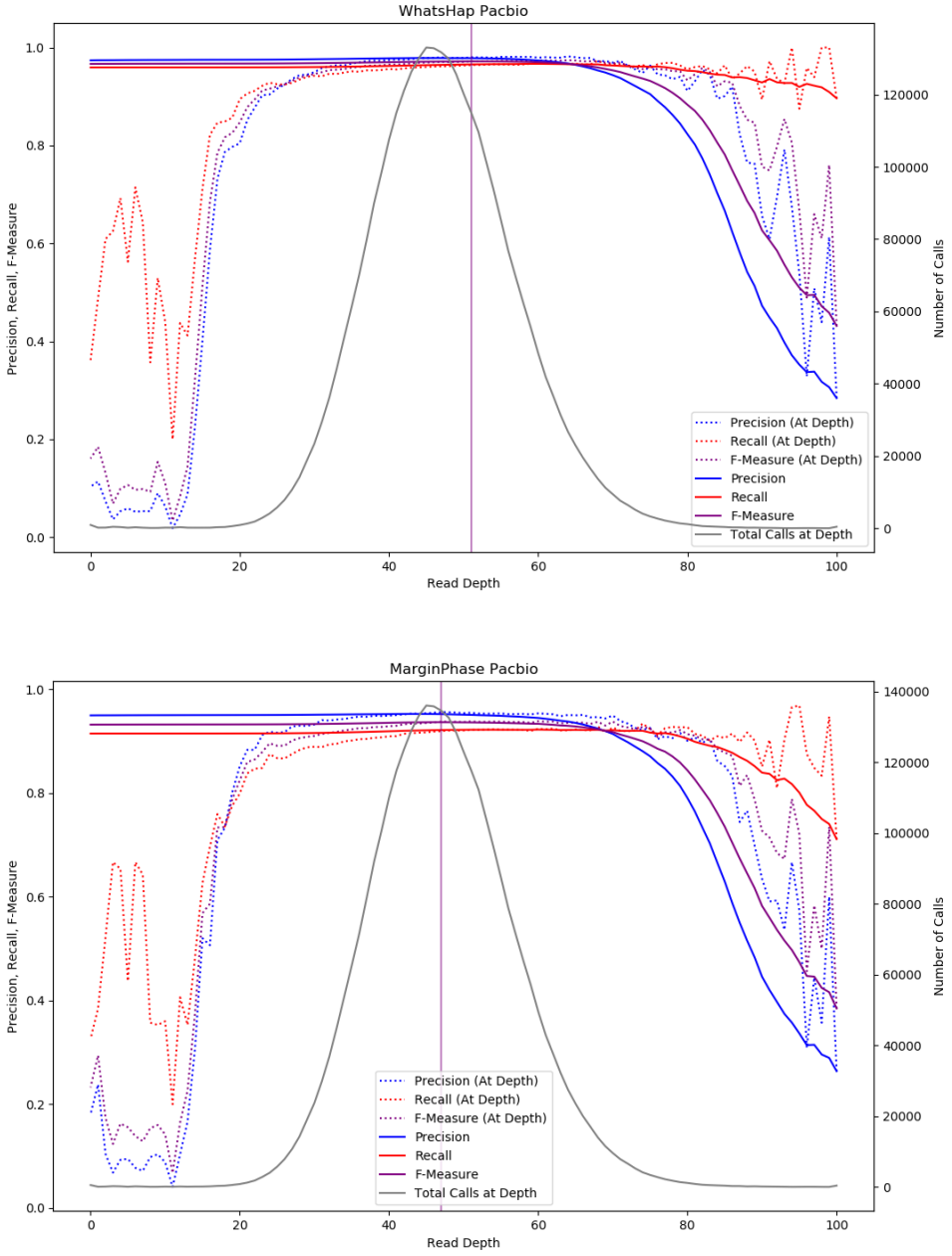

Figure B.3: **Read Depth: PacBio** Precision, Recall, and F-Measure as a function of depth.

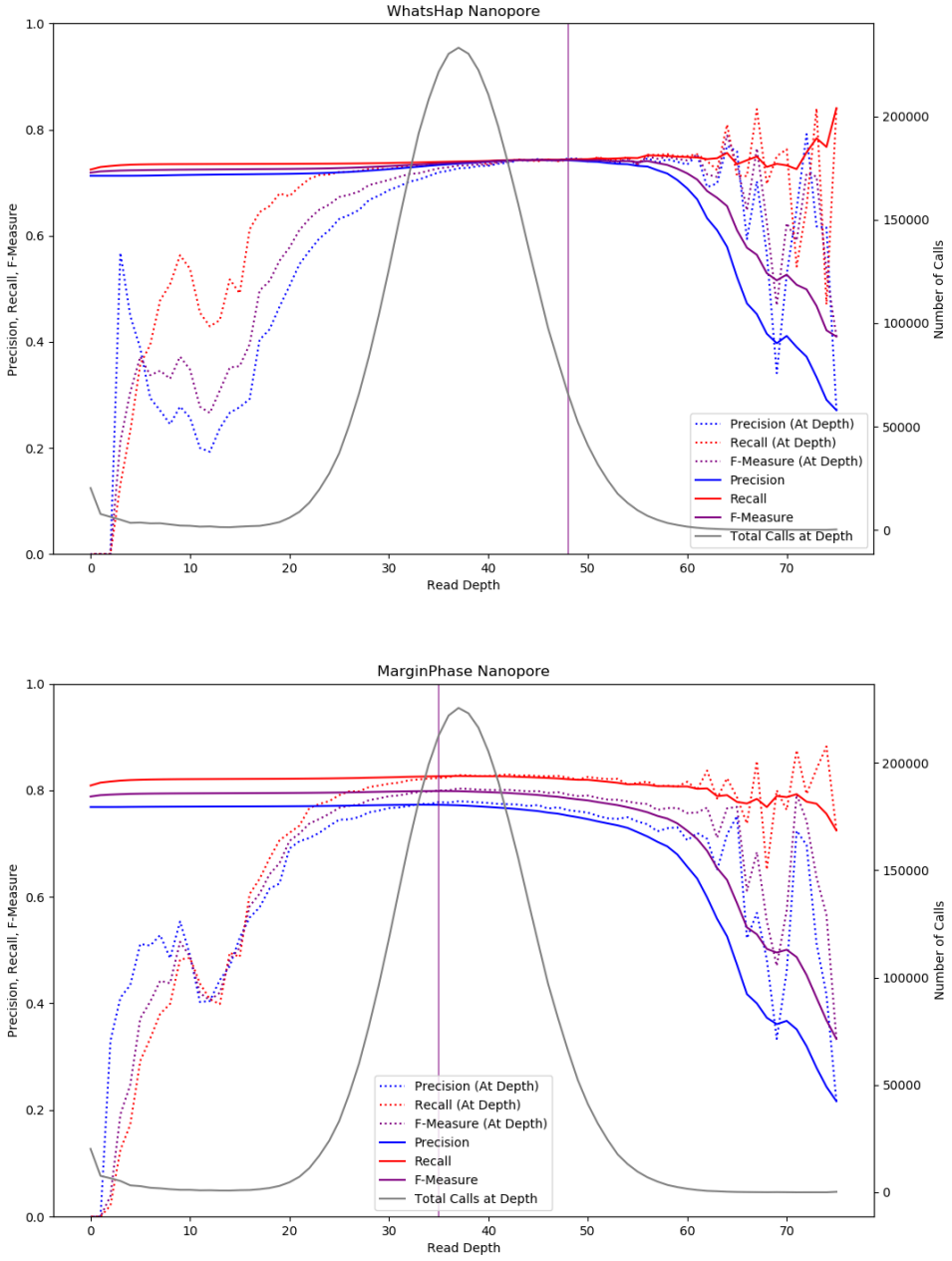

Figure B.4: **Read Depth: Nanopore** Precision, Recall, and F-Measure as a function of depth.

Genotyping Results on Heterozygous Variants vs Homozygous Variants

We present our genotyping results on the Genome in a Bottle truth set in high confidence regions, splitting up the performance on variants that were heterozygous and variants that were homozygous alternate in the truth set. The results are shown in Tables B.3 and B.4. In summary, the precision is better at homozygous sites for all cases (using both tools MarginPhase and WhatsHap, on both PacBio and Nanopore reads). Recall is also better at homozygous sites in most cases, except in the MarginPhase-PacBio run, where it is worse. The difference in performance between heterozygous and homozygous sites is quite drastic when nanopore sequencing is used, especially in regards to precision. Perhaps this means that the programs are predicting many more false variants due to the distribution of errors seen in highly inaccurate reads, and the error models are not yet tuned well enough to take that into account.

	Precision	**Recall**	**F-Measure**
WhatsHap (PacBio)	0.9827	0.9928	0.9877
WhatsHap (Nanopore)	0.9369	0.9382	0.9376
MarginPhase (PacBio)	0.9940	0.8923	0.9404
MarginPhase (Nanopore)	0.9923	0.8448	0.9126

Table B.3: Summary of genotyping results on homozygous variants.

	Precision	**Recall**	**F-Measure**
WhatsHap (PacBio)	0.9678	0.9377	0.9525
WhatsHap (Nanopore)	0.5721	0.5870	0.5795
MarginPhase (PacBio)	0.9241	0.9291	0.9266
MarginPhase (Nanopore)	0.6647	0.7858	0.7202

Table B.4: Summary of genotyping results on heterozygous variants.

Genotype Likelihoods

Our methods output a likelihood for each possible genotype at a variant site and makes a prediction by reporting the likeliest genotype at each position. From the genotype likelihoods, we compute the probability that the reported genotype is wrong by subtracting the likelihood of the predicted genotype from 1. Computing the corresponding phred-score of this value yields the genotype quality.

In order to analyze the reported genotype qualities, we first computed the genotyping concordance of our PacBio and Nanopore callsets with respect to the GIAB truth set as a function of the amount of genotyped variants when using different thresholds on the genotype quality (Figure B.5). For each threshold value (0, 20, 50, 80, 100, 150, 200, 300, 400, 500) we considered the percentage of variants reported with a higher quality score ("variants genotyped") and computed the genotype concordance of this set of variants. Each dot in the plots represents a different threshold, in ascending order from right to left. As it can be seen in Figure B.5, higher thresholds on the genotype quality lead to smaller amounts of genotyped variants. At the same time, the genotype concordance increases since many wrong, low confidence calls are removed. The maximum quality value output by MarginPhase is limited to 100. Therefore, the set of variants genotyped with higher thresholds is empty and no genotype concordance can be computed.

In a second experiment, we compared the genotype concordance of each set of calls reported with the same genotype quality to the expected genotype concordance

as given by the respective qualities. Resulting plots are shown in Figure B.6. The size of each dot corresponds to the number of calls that were reported with the underlying quality in the respective VCF file. Both methods reported high quality values for the majority of calls and the observed genotype concordances for these variants were close to the expected ones. However, plots show that the genotype qualities produced by WhatsHap and MarginPhase are not yet well-calibrated. We expect that improving the computation of weights that we assign to the entries of the allele matrix will lead to better quality scores as the computation of forward and backward probabilities is based on these weights (see Section 5 in the main paper).

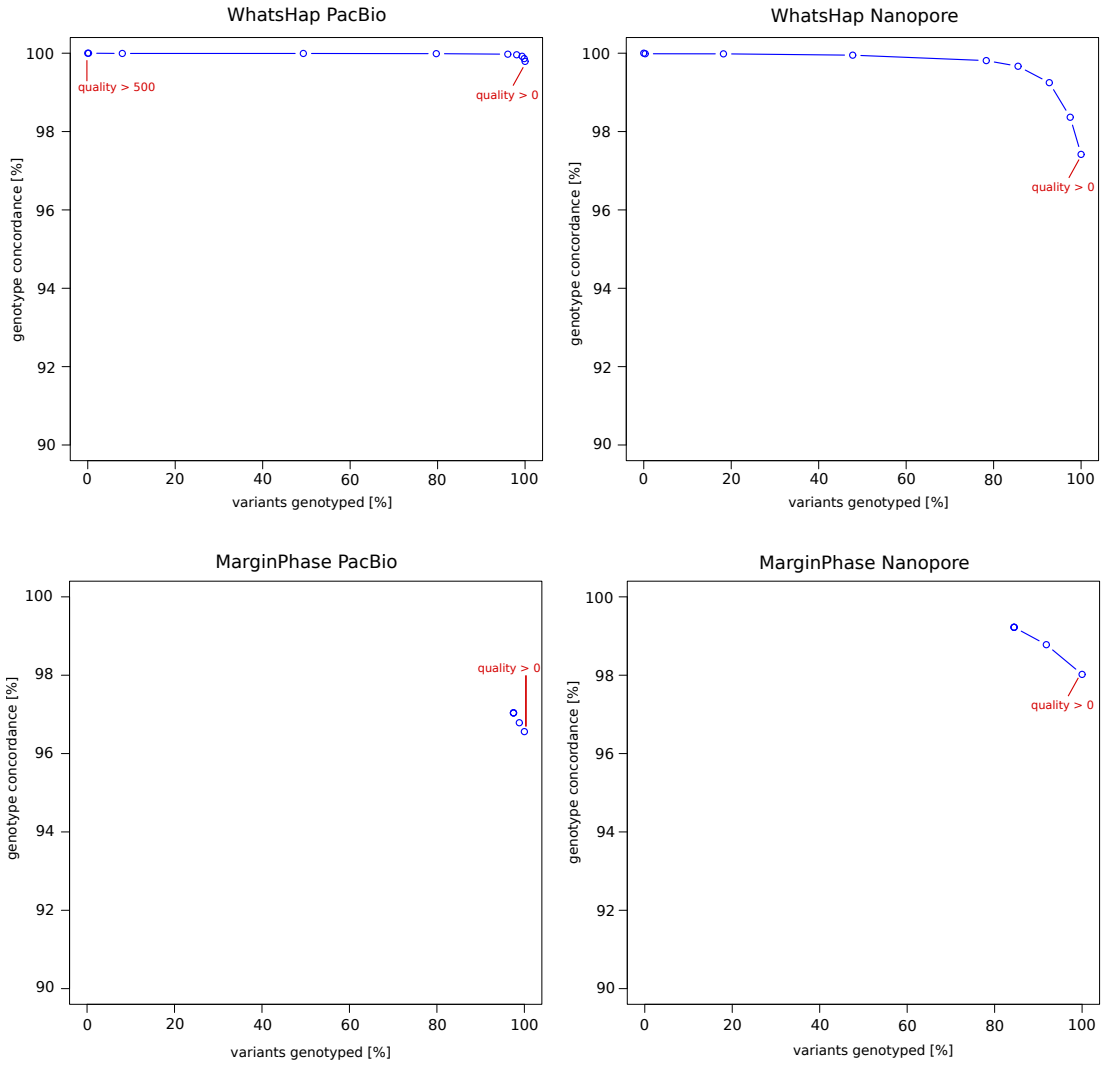

Figure B.5: Genotyping concordance as a function of genotyped variants for different thresholds (0, 20, 50, 80, 100, 150, 200, 300, 400, 500) on the genotype quality. Since the maximum quality value output by MarginPhase is limited to 100, thresholds larger than 100 are not considered in the plots.

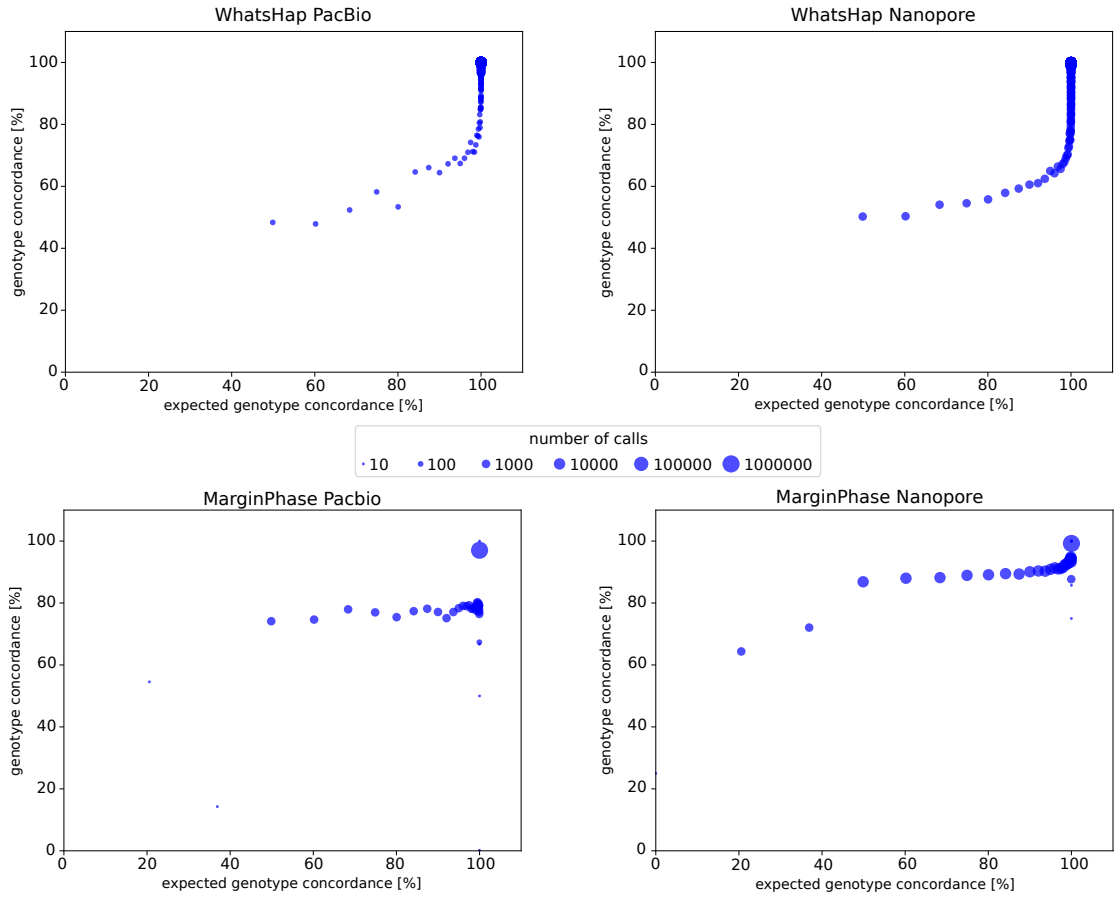

Figure B.6: Observed genotype concordance as a function of the expected genotype concordance of the variant calls. Dot sizes correspond to the number of calls which were reported with the same quality score.

Availability of data and material

The datasets generated and analyzed during the current study as well as the version of the source code used are available at `http://doi.org/10.5281/zenodo.2616973` MarginPhase and WhatsHap are released as Open Source software under the

MIT licence. MarginPhase is available at `github.com/benedictpaten/marginPhase`, WhatsHap is available at `bitbucket.org/whatshap/whatshap`.

Appendix C

Shasta methods

This chapter includes additional methods from the Shasta paper that were omitted from the portion I included from the main text of the paper.

Sample selection

The goal of sample selection was to select a set of individuals that collectively captured the maximum amount of weighted allelic diversity [128]. To do this, we created a list of all low-passage lymphoblastoid cell lines that are part of a trio available from the 1000 Genomes Project collection [28] (We selected trios to allow future addition of pedigree information, and low-passage line to minimize acquired variation). In some cases, we considered the union of parental alleles in the trios due to not having genotypes for the offspring. Let a weighted allele be a variant allele and its frequency in the 1000 Genomes Project Phase 3 VCF. We selected the first sample from our list that contained the largest sum of frequencies of weighted alleles, reasoning that this sample should have

the largest expected fraction of variant alleles in common with any other randomly chosen sample. We then removed the variant alleles from this first sample from the set of variant alleles in consideration and repeated the process to pick the second sample, repeating the process recursively until we had selected seven samples. This set greedily, heuristically optimizes the maximum sum of weighted allele frequencies in our chosen sample subset. We also added the three Ashkenazim Trio samples and the Puerto Rican individual (HG00733). These four samples were added for the purposes of comparison with other studies that are using them [40].

Cell culture

Lymphoblastoid cultures for each individual were obtained from the Coriell Institute Cell Repository (`coriell.org`) and were cultured in RPMI 1640 supplemented with 15% fetal bovine serum (Life Technologies). The cells underwent a total of six passages (p3+3). After expansion, cells were harvested by pelleting at 300xg for 5 minutes. Cells were resuspended in 10 ml PBS and a cell count was taken using a BiRad TC20 cell counter. Cells were aliquoted into 50 ml conical tubes containing 50 million cells, pelleted as above and washed with 10 ml PBS before a final pelleting after which the PBS was removed and the samples were flash frozen on dry ice and stored at -80^{o}C until ready for further processing.

DNA extraction and size-selection

We extracted high-molecular weight (HMW) DNA using the QIAGEN Puregene kit. We followed the standard protocol with some modifications. Briefly, we lysed the cells by adding 3 ml of Cell Lysis Solution per 10 million cells, followed by incubation at 37°C for up to 1 hour. We performed mild shaking intermediately by hand, and avoided vortexing. Once clear, we split the lysate into 3 ml aliquots and added 1 ml of Protein Precipitation Solution to each of the tubes. This was followed by pulse vortexing three times for five seconds each time. We next spun this at 2000 x g for 10 minutes. We added the supernatant from each tube to a new tube containing 3 ml of isopropanol, followed by 50x inversion. The HMW DNA precipitated and formed a dense thread-like jelly. We used a disposable inoculation loop to extract the DNA precipitate. We then dipped the DNA precipitate, while it was on the loop, into ice-cold 70% ethanol. After this, the DNA precipitate was added to a new tube containing 50-250 μl 1x TE buffer. The tubes were heated at 50°C for 2 hours and then left at room temperature overnight to allow resuspension of the DNA. The DNA was then quantified using Qubit and NanoDrop.

We used the Circulomics Short Read Eliminator (SRE) kit to deplete short-fragments from the DNA preparation. We size-selected 10 μg of DNA using the Circulomics recommended protocol for each round of size-selection.

Nanopore sequencing

We used the SQK-LSK109 kit and its recommended protocol for making sequencing libraries. We used 1 μg of input DNA per library. We prepared libraries at a 3x scale since we performed a nuclease flush on every flow cell, followed by the addition of a fresh library.

We used the standard PromethION scripts for sequencing. At around 24 hours, we performed a nuclease flush using the ONT recommended protocol. We then re-primed the flow cell, and added a fresh library corresponding to the same sample. After the first nuclease flush, we restarted the run setting the voltage to -190 mV. We repeated the nuclease flush after another around 24 hours (i.e. around 48 hours into sequencing), re-primed the flow cell, added a fresh library, and restarted the run setting the run voltage to -200 mV.

We performed basecalling using Guppy v.2.3.5 on the PromethION tower using the GPUs. We used the MinION DNA flipflop model (`dna_r9.4.1_450bps_flipflop.cfg`), as recommended by ONT.

Chromatin Crosslinking and Extraction from Human Cell Lines

We thawed the frozen cell pellets and washed them twice with cold PBS before resuspension in the same buffer. We transferred Aliquots containing five million cells by volume from these suspensions to separate microcentrifuge tubes before chromatin crosslinking by addition of paraformaldehyde (EMS Cat. No. 15714) to a final concentration of one percent. We briefly vortexed the samples and allowed them to

incubate at room temperature for fifteen minutes. We pelleted the crosslinked cells and washed them twice with cold PBS before thoroughly resuspending in lysis buffer (50 mM Tris-HCl, 50 mM NaCl, 1 mM EDTA, 1% SDS) to extract crosslinked chromatin.

The Hi-C Method

We bound the crosslinked chromatin samples to SPRI beads, washed three times with SPRI wash buffer (10 mM Tris-HCl, 50 mM NaCl, 0.05% Tween-20), and digested by DpnII (20 U, NEB Catalog No. R0543S) for 1 hour at 37°C in an agitating thermal mixer. We washed the bead-bound samples again before incorporation of Biotin-11-dCTP (ChemCyte Catalog No. CC-6002-1) by DNA Polymerase I, Klenow Fragment (10 U, NEB Catalog No. M0210L) for thirty minutes at 25°C with shaking. Following another wash, we carried out blunt-end ligation by T4 DNA Ligase (4000 U, NEB Catalog No. M0202T) with shaking overnight at 16°C. We reversed the chromatin crosslinks, digested the proteins, eluted the samples by incubation in crosslink reversal buffer (5 mM CaCl 2 , 50 mM Tris-HCl, 8% SDS) with Proteinase K (30 μg, Qiagen Catalog No. 19133) for fifteen minutes at 55°C followed by forty-five minutes at 68°C.

Sonication and Illumina Library Generation with Biotin Enrichment

After SPRI bead purification of the crosslink-reversed samples, we transferred DNA from each to Covaris® microTUBE AFA Fiber Snap-Cap tubes (Covaris Cat. No. 520045) and sonicated to an average length of 400 ± 85 bp using a Covaris® ME220 Focused-Ultrasonicator™. Temperature was held stably at 6°C and treatment

lasted sixty-five seconds per sample with a peak power of fifty watts, ten percent duty factor, and two-hundred cycles per burst. The average fragment length and distribution of sheared DNA was determined by capillary electrophoresis using an Agilent® FragmentAnalyzer 5200 and HS NGS Fragment Kit (Agilent Cat. No. DNF-474-0500). We ran sheared DNA samples twice through the NEBNext® Ultra™ II DNA Library Prep Kit for Illumina® (Catalog No. E7645S) End Preparation and Adaptor Ligation steps with custom Y-adaptors to produce library preparation replicates. We purified ligation products via SPRI beads before Biotin enrichment using Dynabeads® MyOne™ Streptavidin C1 beads (ThermoFisher Catalog No. 65002). We performed indexing PCR on streptavidin beads using KAPA HiFi HotStart ReadyMix (Catalog No. KK2602) and PCR products were isolated by SPRI bead purification. We quantified the libraries by Qubit™ 4 fluorometer and FragmentAnalyzer 5200 HS NGS Fragment Kit (Agilent Cat. No. DNF-474-0500) before pooling for sequencing on an Illumina HiSeq X at Fulgent Genetics.

Analysis methods

Read alignment identities

To generate the identity violin plots (Fig. 3.2c/e) we aligned all the reads for each sample and flowcell to GRCh38 using `minimap2` [79] with the `map-ont` preset. Using a custom script `get_summary_stats.py` in the repository `https://github.com/rlorigro/nanopore_assembly_and_polishing_assessment`, we parsed the alignment for each read and enumerated the number of matched ($N_{=}$), mismatched (N_X), inserted

(N_I), and deleted (N_D) bases. From this, we calculated *alignment identity* as $N_=/(N_= + N_X + N_I + N_D)$. These identities were aggregated over samples and plotted using the `seaborn` library with the script `plot_summary_stats.py` in the same repository. This method was used to generate both Figure 3.2c and Figure 3.2e. For Figure 3.2e, we selected reads from HG00733 flowcell1 aligned to GRCh38 chr1. The "Standard" identities are used from the original reads/alignments. To generate identity data for the "RLE" portion, we extracted the reads above, run-length encoded the reads and chr1 reference, and followed the alignment and identity calculation process described above. Sequences were run-length encoded using a simple script (`github.com/rlorigro/runlength_analysis/blob/master/runlength_encode_fasta.py`) and aligned with minimap2 using the `map-ont` preset and `--k 19`.

Base-level error-rate analysis with Pomoxis

We analyzed the base-level error-rates of the assemblies using the `assess_assembly` tool of Pomoxis toolkit developed by Oxford Nanopore Technology `https://github.com/nanoporetech/pomoxis`. We further modified the program to avoid large insertions and deletions (¿50bp) and submitted a merge request. `https://github.com/nanoporetech/pomoxis/pull/37`. The assess assembly tool is tailored to compute the error rates in a given assembly compared to a truth assembly. It reports an identity error rate, insertion error rate, deletion error rate, and an overall error rate. The identity error rate indicates the number of erroneous substitutions, the insertion error rate is the number of incorrect insertions, and the deletion error rate is the number of deleted

bases averaged over the total aligned length of the assembly to the truth. The overall error rate is the sum of the identity, insertion, and deletion error rates. For the purpose of simplification, we used the indel error rate, which is the sum of insertion and deletion error rates.

The `assess_assembly` script takes an input assembly and a reference assembly to compare against. The assessment tool chunks the reference assembly to 1 Kb regions and aligns it back to the input assembly to get a trimmed reference. Next, the input is aligned to the trimmed reference sequence with the same alignment parameters to get an input assembly to the reference assembly alignment. The total aligned length is the sum of the lengths of the trimmed reference segments where the input assembly has an alignment. The total aligned length is used as the denominator while averaging each of the error categories to limit the assessment in only correctly assembled regions. Then the tool uses `stats_from_bam`, which counts the number of mismatch bases, insert bases, and delete bases at each of the aligned segments and reports the error rate by averaging them over the total aligned length.

The Pomoxis section in Supplementary Notes describe the commands we ran to perform this assessment.

Truth assemblies for base-level error-rate analysis

We used HG002, HG00733, and CHM13 for base-level error-rate assessment of the assembler and the polisher. These three assemblies have high-quality assemblies publicly available, which are used as the ground truth for comparison. Two of the

samples, HG002 and HG00733, are diploid samples; hence, we picked one of the two possible haplotypes as the truth. The reported error rate of HG002 and HG00733 include some errors arising due to the zygosity of the samples. The complete hydatidiform mole sample CHM13 is a haploid human genome which is used to assess the applicability of the tools on haploid samples. We have gathered and uploaded all the files we used for assessment in one place: `https://console.cloud.google.com/storage/browser/kishwar-helen/truth_assemblies/`.

To generate the HG002 truth assembly, we gathered the publicly available Genome-in-a-bottle (GIAB) high-confidence variant set (VCF) against GRCh38 reference sequence. Then we used `bedtools` to create an assembly (FASTA) file from the GRCh38 reference and the high-confidence variant set. We got two files using this process for each of the haplotypes, and we picked one randomly as the truth. All the diploid HG002 assembly is compared against this one chosen assembly. GIAB also provides a bed file annotating high-confidence regions where the called variants are highly precise and sensitive. We used this bed file with `assess_assembly` to ensure that we compare the assemblies only in the high confidence regions.

The HG00733 truth is from the publicly available phased PacBio high-quality assembly of this sample [129]. We picked phase0 as the truth assembly and acquired it from NCBI under accession `GCA_003634895.1`. We note that the assembly is phased but not haplotyped, such that portions of phase0 will include sequences from both parental haplotypes and is not suitable for trio-binned analyses. Furthermore, not all regions were fully phased; regions with variants that are represented as some combination of

both haplotypes will result in lower QV and a less accurate truth.

For CHM13, we used the v0.6 release of CHM13 assembly by the T2T consortium [84]. The reported quality of this truth assembly in Q-value is QV 39. One of the attributes of this assembly is chromosome X. As reported by the T2T assembly authors, chromosome X of CHM13 is the most complete (end-to-end) and high-quality assembly of any human chromosome. We obtained the chromosome X assembly, which is the highest-quality truth assembly (QV $\geq$ 40) we have.

Assembly of MHC

Each of the 8 GRCh38 MHC haplotypes were aligned using `minimap2` (with preset `asm20`) to whole genome assemblies to identify spanning contigs. These contigs were then extracted from the genomic assembly and used for alignment visualization. For dot plots, Nucmer 4.0 [130] was used to align each assembler's spanning contigs to the standard chr6:28000000-34000000 MHC region, which includes ˜500Mb flanks. Output from this alignment was parsed with Dot [131] which has a web-based GUI for visualization. All defaults were used in both generating the input files and drawing the figures. Coverage plots were generated from reads aligned to chr6, using a script, `find_coverage.py`, located at `github.com/rlorigro/nanopore_assembly\_and_polishing_assessment/`.

The best matching alt haplotype (to Shasta, Canu, and Flye) was chosen as a reference haplotype for quantitative analysis. Haplotypes with the fewest supplementary alignments across assemblers were top candidates for QUAST analysis. Candidates with

comparable alignments were differentiated by identity. The highest contiguity/identity MHC haplotype was then analyzed with QUAST using `--min-identity 80`. For all MHC analyses regarding Flye, the unpolished output was used.

BAC Analysis

At a high level, the BAC analysis was performed by aligning BACs to each assembly, quantifying their resolution, and calculating identity statistics on those that were fully resolved.

We obtained 341 BACs for CHM13 [132, 133] and 179 for HG00733 [105] (complete BAC clones of `VMRC62`), which had been selected primarily by targeting complex or highly duplicated regions. We performed the following analysis on the full set of of BACs (for CHM13 and HG00733), and a subset selected to fall within unique regions of the genome. To determine this subset, we selected all BACs which are greater than 10 Kb away from any segmental duplication, resulting in 16 of HG00733 and 31 of CHM13. This subset represents simple regions of the genome which we would expect all assemblers to resolve.

For the analysis, BACs were aligned to each assembly with the command "minimap2 –secondary=no -t 16 -ax asm20 assembly.fasta bac.fasta > assembly.sam" and converted to a PAF-like format which describes aligned regions of the BACs and assemblies. Using this, we calculated two metrics describing how resolved each BAC was: *closed* is defined as having 99.5% of the BAC aligned to a single locus in the assembly; *attempted* is defined as having a set of alignments covering $\geq$ 95% of the

BAC to a single assembly contig where all alignments are at least 1kb away from the contig end. If such a set exists, it counts as attempted. We furthermore calculate median and mean identities (using alignment identity metric described above) of the closed BACs. These definitions were created such that a contig that is counted as attempted but not closed likely reflects a disagreement. The code for this can be found at `https://github.com/skoren/bacValidation`.

Short Read Polishing

Chromosome X of the CHM13 assembly (assembled first with Shasta, then polished with MarginPolish and HELEN) was obtained by aligning the assembly to GRCh38 (using `minimap2` with the `--x asm20` flag). 10X Chromium reads were downloaded from the Nanopore WGS Consortium (`https://github.com/nanopore-wgs-consortium/CHM13/`). These were from a NovaSeq instrument at a coverage of approximately 50X. The reads corresponding to chromosome X were extracted by aligning the entire read set to the whole CHM13 assembly using the 10X Genomics `Long Ranger Align` pipeline (v2.2), then extracting those corresponding to the corresponding chromosome X contigs with `samtools`. Pilon [97] was run iteratively for a total of three rounds, in each round aligning the reads to the current assembly with `Long Ranger` and then running Pilon with default parameters.

Structural Variant Assessment

To create an assembly graph in GFA format Shasta v0.1.0 was run using the HG002 sequence data with `--MarkerGraph.simplifyMaxLength 10` to reduce bubble removal and `--MarkerGraph.highCoverageThreshold 10` to reduce the removal of edges normally removed by the transitive reduction step.

To detect structural variation inside the assembly graphs produced by Shasta, we extracted unitigs from the graph and aligned them back to the linear reference. Unitigs are walks through the assembly graph that do not traverse any node end that includes a bifurcation. We first processed the Shasta assembly graphs (in GFA format) with `gimbricate` (`https://github.com/ekg/gimbricatec1c6d1a`) to recompute overlaps in non run-length encoded space and to remove nodes in the graph only supported by a single sequencing read. To remove overlaps from the graph edges, we then "bluntified" resulting GFAs with `vg find -F` (`https://github.com/vgteam/vgv1.19.0 Tramutola`). We then applied `odgi unitig` (`https://github.com/vgteam/odgi 463ba5b`) to extract unitigs from the graph, with the condition that the starting node in the unitig generation must be at least 100 bp long. To ensure that the unitigs could be mapped back to the linear reference, we appended a random walk of 25 Kb after the natural end of each unitig, with the expectation that even should unitigs would yield around 50 Kb of mappable sequence. Finally, we mapped the unitigs to GRCh38 with `minimap2` with a bandwidth of 25 Kb (`-r25000`), and called variants in the alignments using `paftools.js` from the `minimap2` distribution. We implemented the process in a

single script that produces variant calls from the unitig set of a given graph `https://github.com/ekg/shastaGFA/blob/master/shastaGFAtoVCF_unitig_paftools.sh`.

The extracted variants were compared to the structural variants from the Genome In A Bottle benchmark in HG002 (v0.6, [134]). Precision, recall and F1 scores were computed on variants not overlapping simple repeats and within the benchmark's high-confidence regions. Deletions in the assembly and the GIAB benchmark were matched if they had at least 50% reciprocal overlap. Insertions were matched if located at less than 100 bp from each other and similar in size (50% reciprocal similarity).

Shasta

The following describes Shasta version 0.1.0, which was used throughout our analysis. All runs were done on an AWS `x1.32xlarge` instance (1952 GB memory, 128 virtual processors). The runs used the Shasta recommended options for best performance (`--memoryMode filesystem --memoryBacking 2M`). Rather than using the distributed version of the release, the source code was rebuilt locally for best performance as recommended by Shasta documentation.

Run-length encoding of input reads

Shasta represents input reads using run-length encoding. The sequence of each input read is represented as a sequence of bases, each with a repeat count that says how many times each of the bases is repeated. Such a representation has previously been used in biological sequence analysis [79, 80, 81].

For example, the following read

```
CGATTTAAGTTA
```

is represented as follows using run-length encoding:

```
CGATAGTA
11132121
```

Using run-length encoding makes the assembly process less sensitive to errors in the length of homopolymer runs, which are the most common type of errors in Oxford Nanopore reads. For example, consider these two reads:

```
CGATTTAAGTTA
CGATTAAGGGTTA
```

Using their raw representation above, these reads can be aligned like this:

```
CGATTTAAG--TTA
CGATT-AAGGGTTA
```

Aligning the second read to the first required a deletion and two insertions. But in run-length encoding, the two reads become:

```
CGATAGTA
11132121
CGATAGTA
11122321
```

The sequence portions are now identical and can be aligned trivially and exactly, without any insertions or deletions:

```
CGATAGTA

CGATAGTA
```

The differences between the two reads only appear in the repeat counts:

```
11132121

11122321

  * *
```

The Shasta assembler uses one byte to represent repeat counts, and as a result it only represents repeat counts between 1 and 255. If a read contains more than 255 consecutive bases, it is discarded on input. In the data we have analyzed so far such reads are extremely rare.

Some properties of base sequences in run-length encoding

- In the sequence portion of the run-length encoding, consecutive bases are always distinct. If they were not, the second one would be removed from the run-length encoded sequence, while increasing the repeat count for the first one.

- With ordinary base sequences, the number of distinct k-mers of length k is 4^k. But with run-length base sequences, the number of distinct k-mers of length k is $4 \times 3^{k-1}$. This is a consequence of the previous bullet.

- The run-length sequence is generally shorter than the raw sequence, and cannot be longer. For a long random sequence, the number of bases in the run-length representation is 3/4 of the number of bases in the raw representation.

Markers

Even with run-length encoding, error in input reads are still frequent. To further reduce sensitivity to errors, and also to speed up some of the computational steps in the assembly process, the Shasta assembler also uses a read representation based on *markers*. Markers are occurrences in reads of a pre-determined subset of short k-mers. By default, Shasta uses for this purpose k-mers with $k = 10$ in run-length encoding, corresponding to an average approximately 13 bases in raw read representation.

Just for the purposes of illustration, consider a description using markers of length 3 in run-length encoding. There is a total $4 \times 3^2 = 36$ distinct such markers. We arbitrarily choose the following fixed subset of the 36, and we assign an id to each of the kmers in the subset as follows:

TGC 0

GCA 1

GAC 2

CGC 3

Consider now the following portion of a read in run-length representation (here, the repeat counts are irrelevant and so they are omitted):

```
CGACACGTATGCGCACGCTGCGCTCTGCAGC
```

```
GAC      TGC    CGC        TGC

           CGC     TGC      GCA

             GCA      CGC
```

Occurrences of the k-mers defined in the table above are shown and define the markers in this read. Note that markers can overlap. Using the marker ids defined in the table above, we can summarize the sequence of this read portion as follows:

```
2 0 3 1 3 0 3 0 1
```

This is the marker representation of the read portion above. It just includes the sequence of markers occurring in the read, not their positions.

Note that the marker representation loses information, as it is not possible to reconstruct the complete initial sequence from the marker representation. This also means that the marker representation is insensitive to errors in the sequence portions that don't belong to any markers.

The Shasta assembler uses a random choice of the k-mers to be used as markers. The length of the markers k is controlled by assembly parameter `Kmers.k` with a default value of 10. Each k-mer is randomly choosen to be used as a marker with probability determined by assembly parameter `Kmers.probability` with a default value of 0.1. With these default values, the total number of distinct markers is approximately $0.1 \times 4 \times 3^9 \approx 7900$.

The only constraint used in selecting k-mers to be used as markers is that if a k-mer is a marker, its reverse complement should also be a marker. This makes it easy

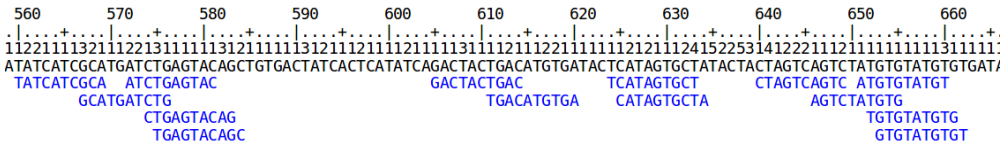

Figure C.1: Markers aligned to a run length encoded read.

to construct the marker representation of the reverse complement of a read from the marker representation of the original read. It also ensures strand symmetry in some of the computational steps.

It is possible that the random selection of markers is not optimal, and that it may be best to select the markers based on their frequency in the input reads or other criteria. These possibilities have not yet been investigated.

Fig. C.1 shows the run-length representation of a portion of a read and its markers, as displayed by the Shasta http server.

Marker alignments

The marker representation of a read is a sequence in an alphabet consisting of the marker ids. This sequence is much shorter than the original sequence of the read, but uses a much larger alphabet. For example, with default Shasta assembly parameters, the marker representation is 10 times shorter than the run-length encoded read sequence, or about 13 times shorter than the raw read sequence. Its alphabet has around 8000 symbols, many more than the 4 symbols that the original read sequence uses.

Because the marker representation of a read is a sequence, we can compute an

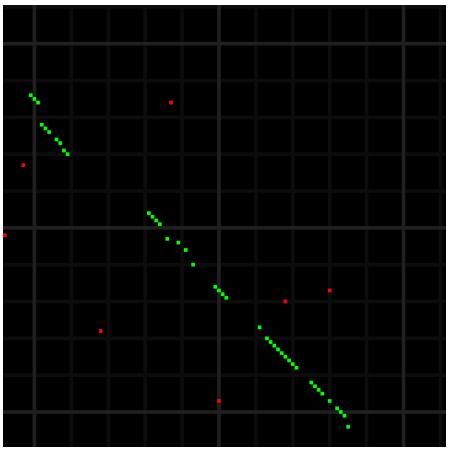

Figure C.2: A marker alignment represented as a dot-plot. Elements that are identical between the two sequences are displayed in green or red - the ones in green are the ones that are part of the optimal alignment computed by the Shasta assembler. Because of the much larger alphabet, matrix elements that are identical between the sequences but are not part of the optimal alignment are infrequent. Each alignment matrix element here corresponds on average to a 13×13 block in the alignment matrix in raw base sequence.

alignment of two reads directly in marker representation. Computing an alignment in this way has two important advantages:

- The shorter sequences and larger alphabet make the alignment much faster to compute.
- The alignment is insensitive to read errors in the portions that are not covered by any marker.

For these reasons, the marker representation is orders of magnitude more efficient than the raw base representation when computing read alignments. Fig. C.2 shows an example alignment matrix.

Computing optimal alignments in marker representation To compute the (likely) optimal alignment (example highlighted in green in Fig. C.2), the Shasta

assembler uses a simple alignment algorithm on the marker representations of the two reads to be aligned. It effectively constructs an optimal path in the alignment matrix, but using some 'banding' heuristics to speed up the computation:

- The maximum number of markers that an alignment can skip on either read is limited to a maximum, under control of assembly parameter `Align.maxSkip` (default value 30 markers, corresponding to around 400 bases when all other Shasta parameters are at their default). This reflects the fact that Oxford Nanopore reads can often have long stretches in error. In the alignment matrix shown in Fig. C.2, there is a skip of about 20 markers (2 light grey squares) following the first 10 aligned markers (green dots) on the top left.

- The maximum number of markers that an alignment can skip at the beginning or end of a read is limited to a maximum, under control of assembly parameter `Align.maxTrim` (default value 30 markers, corresponding to around 400 bases when all other Shasta parameters are at their default). This reflects the fact that Oxford Nanopore reads often have an initial or final portion that is not usable. These first two heuristics are equivalent to computing a reduced band of the alignment matrix.

- To avoid alignment artifacts, marker k-mers that are too frequent in either of the two reads being aligned are not used in the alignment computation. For this purpose, the Shasta assembler uses a criterion based on absolute number of occurrences of marker k-mers in the two reads, although a relative criterion

(occurrences per Kb) may be more appropriate. The current absolute frequency threshold is under control of assembly parameter `Align.maxMarkerFrequency` (default 10 occurrences).

Using these techniques and with the default assembly parameters, the time to compute an optimal alignment is $\sim 10^{-3} - 10^{-2}$ seconds in the Shasta implementation as of release 0.1.0 (April 2019). A typical human assembly needs to compute 10^8 read alignments which results in a total compute time $\sim 10^5 - 10^6$ seconds, or $\sim 10^3 - 10^4$ seconds of elapsed time ($\sim$1-3 hours) on a machine with 128 virtual processors. This is one of the most computationally expensive portions of a Shasta assembly. Some additional optimizations are possible in the code that implement this computation, and may be implemented in future releases.

Finding overlapping reads

Even though computing read alignments in marker representation is fast, it still is not feasible to compute alignments among all possible pairs of reads. For a human size genome with $\sim 10^6 - 10^7$ reads, the number of pairs to consider would be $\sim 10^{12} - 10^{14}$, and even at 10^{-3} seconds per alignment the compute time would be $\sim 10^9 - 10^{11}$ seconds, or $\sim 10^7 - 10^9$ seconds elapsed time ($\sim 10^2 - 10^4$ days) when using 128 virtual processors.

Therefore some means of narrowing down substantially the number of pairs to be considered is essential. The Shasta assembler uses for this purpose a slightly modified MinHash [82, 83] scheme based on the marker representation of reads.

In overview, the MinHash algorithm takes as input a set of items each characterized by a set of features. Its goal is to find pairs of the input items that have a high https://en.wikipedia.org/wiki/Jaccard_indexJaccard similarity index - that is, pairs of items that have many features in common. The algorithm proceeds by iterations. At each iteration, a new hash table is created and a hash function that operates on the feature set is selected. For each item, the hash function of each of its features is evaluated, and the minimum hash function value found is used to select the hash table bucket that each item is stored in. It can be proven that the probability of two items ending up in the same bucket equals the Jaccard similarity index of the two items - that is, items in the same bucket are more likely to be highly similar than items in different buckets [135]. The algorithm then adds to the pairs of potentially similar items all pairs of items that are in the same bucket.

When all iterations are complete, the probability that a pair of items was found at least once is an increasing function of the Jaccard similarity of the two items. In other words, the pairs found are enriched for pairs that have high similarity. One can now consider all the pairs found (hopefully a much smaller set than all possible pairs) and compute the Jaccard similarity index for each, then keep only the pairs for which the index is sufficiently high. The algorithm does not guarantee that all pairs with high similarity will be found - only that the probability of finding all pairs is an increasing function of their similarity.

The algorithm is used by Shasta with items being oriented reads (a read in either original or reverse complemented orientation) and features being consecutive oc-

currences of m markers in the marker representation of the oriented read. For example, consider an oriented read with the following marker representation:

```
18,45,71,3,15,6,21
```

If m is selected equal to 4 (the Shasta default, controlled by assembly parameter `MinHash.m`), the oriented read is assigned the following features:

```
(18,45,71,3)
(45,71,3,15)
(71,3,15,6)
(3,15,6,21)
```

From the picture above of an alignment matrix in marker representation, we see that streaks of 4 or more common consecutive markers are relatively common. We have to keep in mind that, with Shasta default parameters, 4 consecutive markers span an average 40 bases in run-length encoding or about 52 bases in the original raw base representation. At a typical error rate around 10%, such a portion of a read would contain on average 5 errors. Yet, the marker representation in run-length space is sufficiently robust that these common "features" are relatively common despite the high error rate. This indicates that we can expect the MinHash algorithm to be effective in finding pairs of overlapping reads.

However, the MinHash algorithm has a feature that is undesirable for our purposes: namely, that the algorithm is good at finding read pairs with high Jaccard

similarity index. For two sets X and Y, the Jaccard similarity index is defined as the ratio:

$$J = \frac{|X \cap Y|}{|X \cup Y|}$$

Because the read length distribution of Oxford Nanopore reads is very wide, it is very common to have pairs of reads with very different lengths. Consider now two reads with lengths n_x and n_y, with $n_x < n_y$, that overlap exactly over the entire length n_x. The Jaccard similarity is in this case given by $n_x/n_y < 1$. This means that, if one of the reads in a pair is much shorter than the other one, their Jaccard similarity will be low even in the best case of exact overlap. As a result, the unmodified MinHash algorithm will not do a good job at finding overlapping pairs of reads with very different length.

For this reason, the Shasta assembler uses a small modification to the MinHash algorithm: instead of just using the minimum hash for each oriented read for each iteration, it keeps all hashes below a given threshold (this is not the same as keeping a fixed number of the lowest hashes for each read). Each oriented read can be stored in multiple buckets, one for each low hash encountered. The average number of low hashes on a read is proportional to its length, and therefore this change has the effect of eliminating the bias against pairs in which one read is much shorter than the other. The probability of finding a given pair is no longer driver by the Jaccard similarity. The modified algorithm is referred to as *LowHash* in the Shasta source code. Note that it is

effectively equivalent to an indexing approach in which we index all features with low hash.

The LowHash algorithm is controlled by the following assembly parameters:

- `MinHash.m` (default 4): the number of consecutive markers that define a feature.
- `MinHash.hashFraction` (default 0.01): The fraction of hash values that count as "low".
- `MinHash.minHashIterationCount` (default 10): The number of iterations.
- `MinHash.maxBucketSize` (default 10): The maximum number of items for a bucket to be considered. Buckets with more than this number of items are ignored. The goal of this parameter is to mitigate the effect of common repeats, which can result in buckets containing large numbers of unrelated oriented reads.
- `MinHash.minFrequency` (default 2): the number of times a pair of oriented reads has to be found to be considered and stored as a possible pair of overlapping reads.

Initial assembly steps

Initial steps of a Shasta assembly proceed as follows. If the assembly is setup for best performance (`--memoryMode filesystem --memoryBacking 2M` if using the Shasta executable), all data structures are stored in memory, and no disk activity takes place except for initial loading of the input reads, storing of assembly results, and storing a small number of small files with useful summary information.

- Input reads are read from Fasta files and converted to run-length representation.
- K-mers to be used as markers are randomly selected.
- Occurrences of those marker k-mers in all oriented reads are found.
- The LowHash algorithm finds candidate pairs of overlapping oriented reads.
- A marker alignment is computed for each candidate pair of oriented reads. If the marker alignment contains a minimum number of aligned markers, the pair is stored as an aligned pair. The minimum number of aligned markers is controlled by assembly parameter `Align.minAlignedMarkerCount`.

Read graph

Using the methods covered so far, an assembly has created a list of pairs of oriented reads, each pair having a plausible marker alignment. How to use this type of information for assembly is a classical problem with a standard solution `https://doi.org/10.1093/bioinformatics/bti1114`(Myers, 2005), the *string graph*.

It may be possible to adapt the prescriptions in the Myers paper to our situation in which a marker representation is used. However, we have not attempted this here, leaving it for future work. Instead, the approach currently used in the Shasta assembler is very simple, and can likely be improved. In the current simple approach, the Shasta assembler creates un undirected graph, the *Read Graph*, in which each vertex represents an oriented read (that is, a read in either original orientation or reverse complemented), and an undirected edge between two vertices is created if we have found an

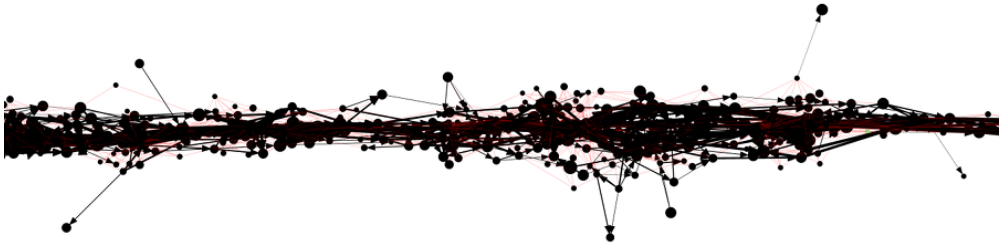

Figure C.3: An example of a portion of the read graph, as displayed by the Shasta http server.

alignment between the corresponding oriented reads.

However, the read graph as constructed in this way suffers from high connectivity in repeat regions. Therefore, the Shasta assembler only keeps a k-Nearest-Neighbor subset of the edges. That is, for each vertex (oriented read) we only keep the k edges with the best alignments (greatest number of aligned markers). The number of edges kept for each vertex is controlled by assembly parameter `ReadGraph.maxAlignmentCount`, with a default value of 6. Note that, despite the k-Nearest-Neighbor subset, it remains possible for a vertex to have degree more than k.

Note that each read contributes two vertices to the read graph, one in its original orientation, and one in reverse complemented orientation. Therefore the read graph contains two strands, each strand at full coverage. This makes it easy to investigate and potentially detect erroneous strand jumps that would be much less obvious if using approaches with one vertex per read.

An example of one strand is shown in Fig. C.3. Even though the graph is undirected, edges that correspond to overlap alignments are drawn with an arrow that points from the prefix oriented read to the suffix one, to represent the direction of

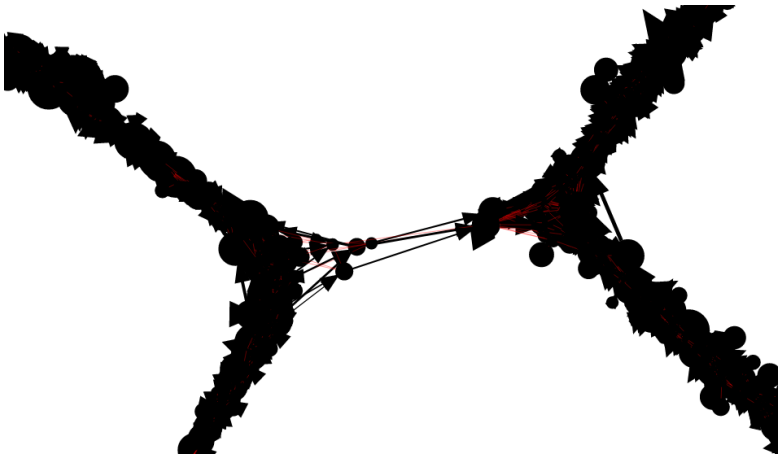

Figure C.4: An example of a portion of the read graph showing obviously incorrect connections

overlap. Edges that correspond to containment alignments (an alignment which covers one of the two reads entirely) are drawn in red and without an arrow. Vertices are drawn with area proportional to the length of the corresponding reads.

The linear structure of the read graph successfully reflects the linear arrangement of the input reads and their origin on the genome being assembled.

However, deviations from the linear structure can occur in the presence of long repeats (Fig. C.4), typically for high similarity segment duplications.

The current Shasta implementation does not attempt to remove the obviously incorrect connections. This results in unnecessary breaks in assembly contiguity. Despite this, Shasta assembly contiguity is adequate and comparable to what other, less performant long read assemblers achieve. It is hoped that future Shasta releases will do a better job at handling these situations.

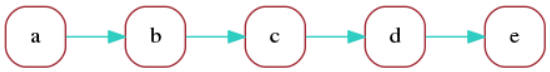

Figure C.5: A marker graph representing a single read.

Marker graph

Consider a read whose marker representation is:

```
a b c d e
```

We can represent this read as a directed graph that the describes the sequence in which its markers appear (Fig. C.5).

This is not very useful but illustrates the simplest form of a *marker graph* as used in the Shasta assembler. The marker graph is a directed graph in which each vertex represents a marker and each edge represents the transition between consecutive markers. We can associate sequence with each vertex and edge of the marker graph:

- Each vertex is associated with the sequence of the corresponding marker.
- If the markers of the source and target vertex of an edge do not overlap, the edge is associated with the sequence intervening between the two markers.
- If the markers of the source and target vertex of an edge do overlap, the edge is associated with the overlapping portion of the marker sequences.

Consider now a second read with the following marker representation, which differs from the previous one just by replacing marker `c` with `x`:

```
a b x d e
```

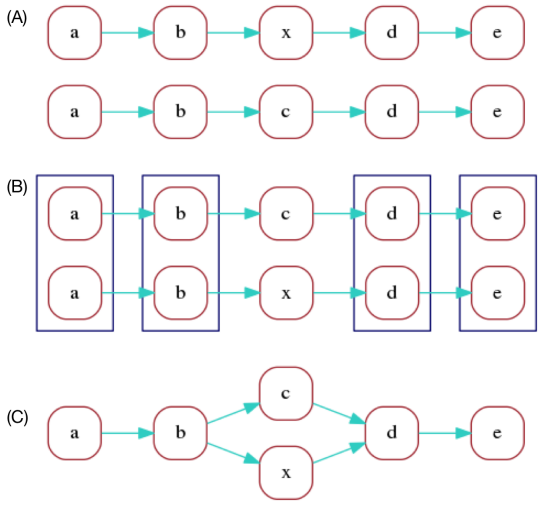

Figure C.6: An illustration of marker graph construction for two sequences.

The marker graph for the two reads is Fig C.6(A).

In the optimal alignment of the two reads, markers `a, b, d, e` are aligned. We can redraw the marker graph grouping together vertices that correspond to aligned markers as in Fig C.6(B).

Finally, we can merge aligned vertices to obtain a marker graph describing the two aligned reads, shown in Fig C.6(C).

Here, by construction each vertex still has a unique sequence associated with it - the common sequence of the markers that were merged (however the corresponding repeat counts can be different for each contributing read). An edge, on the other hand, can have different sequences associated with it, one corresponding to each of the contributing reads. In this example, edges `a->b` and `d->e` have two contributing reads, which can each have distinct sequence between the two markers.

We call coverage of a vertex or edge the number of reads "contributing" to it.

In this example, vertices `a, b, d, e` have coverage 2 and vertices `c, x` have coverage 1. Edges `a->b` and `d->e` have coverage 2, and the remaining edges have coverage 1.

The construction of the marker graph was illustrated above for two reads, but the Shasta assembler constructs a global marker graph which takes into account all oriented reads:

- The process starts with a distinct vertex for each marker of each oriented read. Note that at this stage the marker graph is large ($\sim 2 \times 10^{10}$ vertices for a human assembly using default assembly parameters).
- For each marker alignment corresponding to an edge of the read graph, we merge vertices corresponding to aligned markers.
- Of the resulting merged vertices, we remove those whose coverage in too low or two high, indicating that the contributing reads or some of the alignments involved are probably in error. This is controlled by assembly parameters `MarkerGraph.minCoverage` (default 10) and `MarkerGraph.maxCoverage` (default 100), which specify the minimum and maximum coverage for a vertex to be kept.
- Edges are created. An edge `v0->v1` is created if there is at least a read contributing to both `v0` and `v1` and for which all markers intervening between `v0` and `v1` belong to vertices that were removed.

Note that this does not mean that all vertices with the same marker sequence are merged - two vertices are only merged if they have the same marker sequence, and if there are at least two reads for which the corresponding markers are aligned.

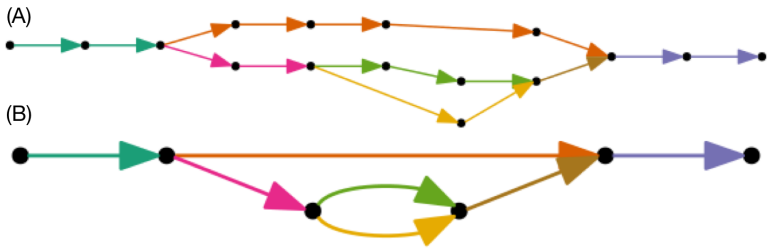

Figure C.7: (A) A marker graph with linear sequence of edges colored. (B) The corresponding assembly graph. Colors were chosen to indicate the correspondence to marker graph edges.

Given the large number of initial vertices involved, this computation is not trivial. To allow efficient computation in parallel on many threads a lock-free implementation of the disjoint data set data structure [136], is used for merging vertices. Some code changes were necessary to permit large numbers of vertices, as the initial implementation by Wenzel Jakob only allowed for 32-bit vertex ids (`https://github.com/wjakob/dset`).

Assembly graph

The Shasta assembly process also uses a compact representation of the marker graph, called the *assembly graph*, in which each linear sequence of edges is replaced by a single edge (Fig. C.7).

The *length* of an edge of the assembly graph is defined as the number of marker graph edges that it corresponds to. For each edge of the assembly graph, an average coverage is also computed, by averaging the coverage of the marker graph edges it corresponds to.

Using the marker graph to assemble sequence

The marker graph is a partial description of the multiple sequence alignment between reads and can be used to assemble consensus sequence. One simple way to do that is to only keep the "dominant" path in the graph, and then traverse that path from vertex to edge to vertex, assembling run-length encoded sequence as follows:

1. On a vertex, all reads have the same sequence, by construction: the marker sequence associated with the vertex. There is trivial consensus among all the reads contributing to a vertex, and the marker sequence can be used directly as the contribution of the vertex to assembled sequence.

2. For edges, there are two possible situations plus a hybrid case:

    - 2.1. If the adjacent markers overlap, in most cases all contributing reads have the same number of overlapping bases between the two markers, and we are again in a situation of trivial consensus, where all reads contribute the same sequence, which also agrees with the sequence of adjacent vertices. In cases where not all reads are in agreement on the number of overlapping bases, only reads with the most frequent number of overlapping bases are taken into account.

    - 2.2. If the adjacent markers don't overlap, then each read can have a different sequence between the two markers. In this situation, we compute a multiple sequence alignment of the sequences and a consensus using the spoa library [92] (`https://github.com/rvaser/spoa`). The multiple sequence alignment

is computed constrained at both ends, because all reads contributing to the edge have, by construction, identical markers at both sides.

- 2.3. A hybrid situation occasionally arises, in which some reads have the two markers overlapping, and some do not. In this case we count reads of the two kinds and discard the reads of the minority kind, then revert to one of the two cases 2.1 or 2.2 above.

This is the process used for sequence assembly by the current Shasta implementation. It requires a process to select and define dominant paths, which is described in the next section. It is algorithmically simple, but its main shortcoming is that it does not use for assembly reads that contribute to the abundant side branches. This means that coverage is lost, and therefore the accuracy of assembled sequence is not as good as it could be if all available coverage was used. Means to eliminate this shortcoming and use information from the side branches of the marker graph could be a subject of future work on the Shasta assembler.

The process described above works with run-length encoded sequence and therefore assembles run-length encoded sequence. The final step to create raw assembled sequence is to compute the most likely repeat count for each sequence position in run-length encoding. After some experimentation, this is currently done by choosing as the most likely repeat count the one that appears the most frequently in the reads that contributed to each assembled position.

A simple Bayesian model for repeat counts resulted in a modest improvement

in the quality of assembled sequence. But the model appears to sensitive to calibration errors, and therefore it is not used by default in Shasta assemblies. However, it is used by MarginPolish, as described below.

Selecting assembly paths in Shasta

The sequence assembly procedure described in the previous section can be used to assemble sequence for any path in the marker graph. This section describes the selection of paths for assembly in the current Shasta implementation. This is done by a series of steps that "remove" edges (but not vertices) from the marker graph until the marker graph consists mainly of linear sections which can be used as the assembly paths. For speed, edges are not actually removed but just marked as removed using a set of flag bits allocated for this purpose in each edge. However, the description below will use the loose term "remove" to indicate that an edge was flagged as removed.

This process consists of the following three steps, described in more detail in the following sections:

- TransitiveReduction: Approximate transitive reduction of the marker graph.
- Pruning:Pruning of short side branches (leaves).
- BubbleRemoval:Removal of bubbles and super-bubbles.

The last step, removal of bubbles and superbubbles, is consistent with Shasta's current assembly goal which is to compute a mostly monoploid assembly, at least on short scales.

TransitiveReduction:

The goal of this step is to eliminate the side branches in the marker graph, which are the result of errors. Despite the fact that the number of side branches is substantially reduced thanks to the use of run-length encoding, side branches are still abundant. This step uses an approximate transitive reduction of the marker graph which only considers reachability up to a maximum distance, controlled by assembly parameter `MarkerGraph.maxDistance` (default 30 marker graph edges). Using a maximum distance makes sure that the process remains computationally affordable, and also has the advantage of not removing long-range edges in the marker graph, which could be significant.

In detail, the process works as follows. In this description, the edge being considered for removal is the edge `v0`→`v1` with source vertex `v0` and target vertex `v1`. The first two steps are not really part of the transitive reduction but are performed by the same code for convenience.

- All edges with coverage less than or equal to `MarkerGraph.lowCoverageThreshold` are unconditionally removed. The default value for this assembly parameter is 0, so this step does nothing when using default parameters.
- All edges with coverage 1 and for which the only supporting read has a large marker skip are unconditionally removed. The marker skip of an edge, for a given read, is defined as the distance (in markers) between the `v0` marker for

that read and the v1 marker for the same read. Most marker skips are small, and a large skip is indicative of an artifact. Keeping those edges could result in assembly errors. The marker skip threshold is controlled by assembly parameter MarkerGraph.edgeMarkerSkipThreshold (default 100 markers).

- Edges with coverage greater than MarkerGraph.lowCoverageThreshold (default 0) and less than MarkerGraph.highCoverageThreshold (default 256), and that were not previously removed, are processed in order of increasing coverage. Note that with the default values of these parameters all edges are processed, because edge coverage is stored using one byte and therefore can never be more than 255 (it is saturated at 255). For each edge v0→v1, a https://en.wikipedia.org/wiki/Breadth-first_searchBreadth-First Search (BFS) in the marker graph is performed starting at source vertex v0 and with a limit of MarkerGraph.maxDistance (default 30) edges distance from vertex v0. The BFS is constrained to not use edge v0→v1. If the BFS reaches v1, indicating that an alternative path from v0 to v1 exists, edge v0→v1 is removed. Note that the BFS does not use edges that have already been removed, and so the process is guaranteed not to affect reachability. Processing edges in order of increasing coverage makes sure that low coverage edges the most likely to be removed.

The transitive reduction step is intrinsically sequential and so it is currently performed in sequential code for simplicity. It could be in principle be parallelized, but that would require sophisticated locking of marker graph edges to make sure independent

threads don't step on each other, possibly reducing reachability. However, even with sequential code, this step is not computationally expensive, taking typically only a small fraction of total assembly time.

When the transitive reduction step is complete, the marker graph consists mostly of linear sections composed of vertices with in-degree and out-degree one, with occasional side branches and bubbles or `https://arxiv.org/abs/1307.7925`superbubbles, which are handled in the next two phases described below.

Pruning

At this stage, a few iterations of pruning are done by simply removing, at each iteration, edge `v0`→`v1` if `v0` has in-degree 0 (that is, is a backward-pointing leaf) or `v1` has out-degree 0 (that is, is a forward-pointing leaf). The net effect is that all side branches of length (number of edges) at most equal to the number of iterations are removed. This leaves the leaf vertex isolated, which causes no problems. The number of iterations is controlled by assembly parameter `MarkerGraph.pruneIterationCount` (default 6).

BubbleRemoval

The marker graph now consists of mostly linear section with occasional bubbles or superbubbles [137]. Most of the bubbles and superbubbles are caused by errors, but some of those are due to heterozygous loci in the genome being assembled. Bubbles and superbubbles of the latter type could be used for separating haplotypes (phasing) - a possibility that will be addressed in future Shasta releases. However, the goal of

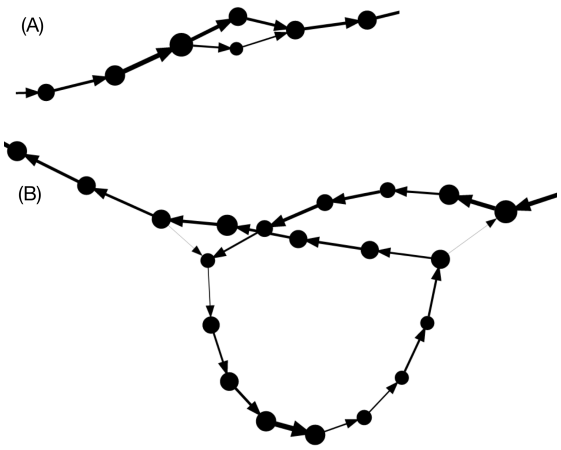

Figure C.8: (A) A simple bubble. (B) A superbubble.

the current Shasta implementation is to create a monoploid assembly at all scales but the very long ones. Accordingly, bubbles and superbubbles at short scales are treated as errors, and the goal of the bubble/superbubble removal step is to keep the most significant path in each bubble or superbubble.

The Fig. C.8 shows typical examples of a bubble and superbubble in the marker graph.

The bubble/superbubble removal process is iterative. Early iterations work on short scales, and late iterations fork on longer scales. Each iteration uses a length threshold that controls the maximum number of marker graph edges for features to be considered for removal. The value of the threshold for each iteration is specified using assembly parameter `MarkerGraph.simplifyMaxLength`, which consists of a comma-separated string of integer numbers, each specifying the threshold for one iteration in the process. The default value is `10,100,1000`, which means that three iterations of this

process are performed. The first iteration uses a threshold of 10 marker graph edges, and the second and third iterations use length thresholds of 100 and 1000 marker graph edges, respectively. The last and largest of the threshold values used determines the size of the smallest bubble or superbubble that will survive the process. The default 1000 markers is equivalent to roughly 13 Kb. To suppress more bubble/superbubbles, increase the threshold for the last iteration. To see more bubbles/superbubbles, decrease the length threshold for the last iteration, or remove the last iteration entirely.

The goal of the increasing threshold values is to work on small features at first, and on larger features in the later iterations. The choice of `MarkerGraph.simplifyMaxLength` could be application dependent. The default value is a reasonable compromise useful if one desires a mostly monoploid assembly with just some large heterozygous features.

Each iteration consists of two steps. The first removes bubbles and the second removes superbubbles. Only bubbles/superbubbles consisting of features shorter than the threshold for the current iteration are considered:

1. **Bubble removal**

    - An assembly graph corresponding to the current marker graph is created.
    - Bubbles are located in which the length of all branches (number of marker graph edges) is no more than the length threshold at the current iteration. In the assembly graph, a bubble appears as a set of parallel edges (edges with

the same source and target).

- In each bubble, only the assembly graph edge with the highest average coverage is kept. Marker graph edges corresponding to all other assembly graph edges in the bubble are flagged as removed.

2. **Superbubble removal**:

- An assembly graph corresponding to the current marker graph is created.
- Connected components of the assembly graph are computed, but only considering edges below the current length threshold. This way, each connected component corresponds to a "cluster" of "short" assembly graph edges.
- For each cluster, entries in the cluster are located. These are vertices that have in-edges from a vertex outside the cluster. Similarly, out-edges are located (vertices that have out-edges outside the cluster).
- For each entry/exit pair, the shortest path is computed. However, in this case the "length" of an assembly graph edge is defined as the inverse of its average coverage - that is, the inverse of average coverage for all the contributing marker graph edges.
- Edges on each shortest path are marked as edges to be kept.
- All other edges internal to the cluster are removed.

When all iterations of bubble/superbubble removal are complete, the assem-

bler creates a final version of the assembly graph. Each edge of the assembly graph corresponds to a path in the marker graph, for which sequence can be assembled using the assembleSequence method described above. Note, however, that the marker graph and the assembly graph have been constructed to contain both strands. Special care is taken during all transformation steps to make sure that the marker graph (and therefore the assembly graph) remain symmetric with respect to strand swaps. Therefore, the majority of assembly graph edges come in reverse complemented pairs, of which we assemble only one. It is however possible but rare for an assembly graph to be its own reverse complement.

Assembly parameters selection The sequence of computational steps outlined above depends on a number of assembly parameters, like for example the length and fraction of k-mers used as markers, the parameters controlling the LowHash iteration, and so on. In Shasta, all of these parameters are exposed as command line options and none of them are hardcoded or hidden. Our error analysis shows that the set of assembly parameters we used (the default values for Shasta 0.1.0) gave satisfactory assembly results for our data. However we do not claim that the same choices would generalize to other situations. Additional work will be needed to find parameter sets that work for lower or higher coverage, for genomes of different sizes and characteristics, or for different types of long reads.

High performance computing techniques employed by Shasta The Shasta assembler is designed to run on a single machine with an amount of memory sufficient to hold all of its data structures (1-2 TB for a human assembly, depending on coverage). All data structures are memory mapped and can be set up to remain available after assembly completes. Note that using such a large memory machine does not substantially increase the cost per CPU cycle. For example, on Amazon AWS the cost per virtual processor hour for large memory instances is no more than twice the cost for laptop-sized instances.

There are various advantages to running assemblies in this way:

- Running on a single machine simplifies the logistics of running an assembly, versus for example running on a cluster of smaller machines with shared storage.
- No disk input/output takes place during assembly, except for loading the reads in memory and writing out assembly results plus a few small files containing summary information. This eliminates performance bottlenecks commonly caused by disk I/O.
- Having all data structures in memory makes it easier and more efficient to exploit parallelism, even at very low granularity.
- Algorithm development is easier, as all data are immediately accessible without the need to read files from disk. For example, it is possible to easily rerun a specific portion of an assembly for experimentation and debugging without any wait time for data structures to be read from disk.

- When the assembler data structures are set up to remain in memory after the assembler completes, `InspectingResults.html`it is possible to use the Python API or the Shasta http server to inspect and analyze an assembly and its data structures (for example, display a portion of the read graph, marker graph, or assembly graph).

- For optimal performance, assembler data structures can be mapped to Linux 2 MB pages ("huge pages"). This makes it faster for the operating system to allocate and manage the memory, and improves TLB efficiency. Using huge pages mapped on the `hugetlbfs` filesystem (Shasta executable options `--memoryMode filesystem --memoryBacking 2M`) can result in a significant speed up (20-30%) for large assemblies. However it requires root privilege via `sudo`.

To optimize performance in this setting, the Shasta assembler uses various techniques:

- In most parallel steps, the division of work among threads is not set up in advance but decided dynamically ("Dynamic load balancing"). As a thread finishes a piece of work assigned to it, it grabs another chunk of work to do. The process of assigning work items to threads is lock-free (that is, it uses atomic memory primitives rather than mutexes or other synchronization methods provided by the operating system).

- Most large memory allocations are done via `mmap` and can optionally be mapped to Linux 2 MB pages backed by the Linux `hugetlbfs`. This memory is persistent

until the next reboot and is resident (non-pageable). As a result, assembler data structures can be kept in memory and reaccessed repeatedly at very low cost. This facilitates algorithm development (e. g. it allows repeatedly testing a single assembly phase without having to rerun the entire assembly each time or having to wait for data to load) and postprocessing (inspecting assembly data structures after the assembly is complete). The Shasta http server and Python API take advantage of this capability.

- The Shasta code includes a C++ class (class shasta::MemoryMapped::Vector) for conveniently handling these large memory-mapped regions as C++ containers with familiar semantics.

- In situations where a large number of small vectors are required, a two-pass process is used (class shasta::MemoryMapped::VectorOfVectors). In the first pass, one computes the length of each of the vectors. A single large area is then allocated to hold all of the vectors contiguously, together with another area to hold indexes pointing to the beginning of each of the short vectors. In a second pass, the vectors are then filled. Both passes can be performed in parallel and are entirely lock-free. This process eliminates memory allocation overhead that would be incurred if each of the vectors were to be allocated individually.

Thanks to these techniques, Shasta achieves close to 100% CPU utilization during its parallel phases, even when using large numbers of threads. However, a number of sequential phases remain, which typically result in average CPU utilization during a

large assembly around 70%. Some of these sequential phases can be parallelized, which would result in increased average CPU utilization and improved assembly performance.

MarginPolish

Throughout we used MarginPolish (`https://github.com/ucsc-nanopore-cgl/MarginPolish`) version 1.0.0.

MarginPolish is an assembly refinement tool designed to sum over (marginalize) read to assembly alignment uncertainty. It takes as input a genome assembly and set of aligned reads in BAM format.

It outputs a refined version of the input genome assembly after attempting to correct base-level errors in terms of substitutions and indels (insertions and deletions). It can also output a summary representation of the assembly and read alignments as a weighted partial order alignment graph (POA), which is used by the HELEN neural network based polisher described below.

It was designed and is optimized to work with noisy long ONT reads, although parameterization for other, similar read types is easily possible. It does not yet consider signal-level information from ONT reads. It is also currently a haploid polisher, in that it does not attempt to recognize or represent heterozygous polymorphisms or phasing relationships. For haploid genome assemblies of a diploid genome it will therefore fail to capture half of all heterozygous polymorphisms.

Algorithm Overview MarginPolish works in overview as follows:

1. Reads and the input assembly are converted to their run-length encoding (RLE) (see Shasta description above for description and rationale).

2. A restricted, weighted Partial Order Alignment [92] (POA) graph is constructed representing the RLE input assembly and potential edits to it in terms of substitutions and indels.

3. Within identified regions of the POA containing likely assembly errors:

    - A set of alternative sequences representing combinations of edits are enumerated by locally traversing the POA within the region.
    - The likelihood of the existing and each alternative sequence is evaluated given the aligned reads.
    - If an alternative sequence with higher likelihood than the current reference exists then the assembly at the location is updated with this higher likelihood sequence.

4. Optionally, the program loops back to step 2 to repeat the refinement process (by default it loops back once).

5. The modified RLE assembly is expanded by estimating the repeat count of each base given the reads using a simple Bayesian model. The resulting final, polished assembly is output. In addition, a representation of the weighted POA can be output.

Innovations Compared to existing tools MarginPolish is most similar to Racon [94], in that they are comparable in speed, both principally use small-parameter HMM like models and both do not currently use signal information. Compared to Racon MarginPolish has some key innovations that we have found to improve polishing accuracy:

- MarginPolish, as with our earlier tool in the Margin series [2], uses the forward-backward and forward algorithms for pair hidden Markov models (HMMs) to sum over all possible pairwise alignments between pairs of sequences instead of the single most probable alignment (Viterbi). Considering all alignments allows more information to be extracted per read.

- The POA graph is constructed from a set of weights computed from the posterior alignment probabilities of each read to the initial assembled reference sequence (see below), the result is that MarginPolish POA construction does not have a read-order dependence. This is somewhat similar to that described by HGAP3 [138]. Most earlier algorithms for constructing POA graphs have a well known explicit read order dependence that can result in undesirable topologies [92].

- MarginPolish works in run-length encoded space, which results in considerably less alignment uncertainty and correspondingly improved performance.

- MarginPolish, similarly to Nanopolish [139], evaluates the likelihood of each alternative sequence introduced into the assembly. This improves performance relative to a faster but less accurate algorithm that traces back a consensus sequence

through the POA graph.

- MarginPolish employs a simple chunking scheme to break up the polishing of the assembly into overlapping pieces. This results in low memory usage per core and simple parallelism.

Below steps 2, 3 and 5 of the MarginPolish algorithm are described in detail. In addition, the parallelization scheme is described.

Partial Order Alignment Graph Construction To create the POA we start with the existing assembled sequence $s = s_1, s_2, \ldots s_n$ and for each read $r = r_1, r_2, \ldots, r_m$ in the set of reads R use the Forward-Backward algorithm with a standard 3-state, affine-gap pair-HMM to derive posterior alignment probabilities using the implementation described in [106]. The parameters for this model are specified in the `polish.hmm` subtree of the JSON formatted parameters file, including "polish.hmm.transitions", and "polish.hmm.emissions". Current defaults were tuned via expectation maximization [70] of R9.4 ONT reads aligned to a bacterial reference; we have observed the parameters for this HMM seem robust to small changes in basecaller versions. The result of running the Forward-backward algorithm is three sets of posterior probabilities:

- Firstly *match probabilities*: the set of posterior match probabilities, each the probability $P(r_i \diamond s_j)$ that a read base r_i is aligned to a base s_j in s.

- Secondly *insertion probabilities*: the set of posterior insertion probabilities, each

the probability $P(r_i \diamond -j)$ that a read base r_i is inserted between two bases s_j and s_{j+1} in s, or, if $j = 0$, inserted before the start of s, or, if $j = n$, after the end of s.

- Thirdly *deletion probabilities*, the set of posterior deletion probabilities, each the probability $P(-i \diamond s_j)$ that a base s_j in s is deleted between two read bases r_i and r_{i+1}. (Note, because a read is generally an incomplete observation of s we consider the probability that a base in s is deleted before the first position or after the last position of a read as 0).

As most probabilities in these three sets are very small and yet to store and compute all the probabilities would require evaluating comparatively large forward and backward alignment matrices we restrict the set of probabilities heuristically as follows:

- We use a banded forward-backward algorithm, as originally described here [140]. To do this we use the original alignment of the read to s as in the input BAM file. Given that s is generally much longer than each read this allows computation of each forward-backward invocation in time linearly proportional to the length of each read, at the cost of restricting the probability computation to a sub-portion of the overall matrix, albeit one that contains the vast majority of the probability mass.

- We only store posterior probabilities above a threshold (`polish.pairwiseAlignmentParameters.threshold`, by default 0.01), treating smaller probabilities as equivalent as zero.

The result is that these three sets of probabilities are a very sparse subset of the complete sets.

To estimate the posterior probability of a multi-base insertion of a read substring $r_i, r_{i+1}, \ldots r_k$ at a given location j in s involves repeated summation over terms in the forward and backward matrices. Instead to approximate this probability we heuristically use:

$$P(r_i, r_{i+1}, \ldots r_k \diamond -j) = \arg\min_{l \in [i,k]} P(r_l \diamond -j)$$

the minimum probability of any base in the multi-base insertion being individually inserted at the location in s as a proxy, a probability that is an upper-bound on the actual probability.

Similarly we estimate the posterior probability of a deletion involving more than one contiguous base s at a given location in a read using analogous logic. As we store a sparse subset of the single-base insertion and deletion probabilities and given these probability approximations it is easy to calculate all the multi-base indel probabilities with value greater than t by linear traversal of the single-based insertion and deletion probabilities after sorting them, respectively, by their read and s coordinates. The result of such calculation is expanded sets of insertion and deletion probabilities that include multi-base probabilities.

To build the POA we start from s, which we call the *backbone*. The backbone is a graph where each base s_j in s corresponds to a node, there are special source and sink nodes (which do not have a base label), and the directed edges connect the nodes

for successive bases s_j, s_{j+1} in s, from the source node to the node for s_1, and, similarly, from the node for s_n to the sink node.

Each non-source/sink node in the backbone has a separate weight for each possible base $x \in \{A, C, G, T\}$. This weight:

$$w(j, x) = \sum_{r \in R} \sum_{i} \mathbb{1}_x(r_i) P(r_i \diamond s_j)$$

where $\mathbb{1}_x(r_i)$ is an indicator function that is 1 if $r_i = x$ and otherwise 0, corresponds to the sum of match probabilities of read elements of base x being aligned to s_j. This weight has a probabilistic interpretation: it is the total number of expected observations of the base x in the reads aligned to s_j, summing over all possible pairwise alignments of the reads to s. It can be fractional because of the inherent uncertainty of these alignments, e.g. we may predict only a 50% probability of observing such a base in a read.

We add *deletion edges*, which connect nodes in the backbone. Indexing the nodes in the backbone from 0 (the source) to the source $n + 1$ (the sink), a deletion edge between positions j and k in the backbone corresponds to the deletion of bases $j, j + 1, \ldots k - 1$ in s. Each deletion edge has a weight equal to the sum of deletion probabilities for deletion events that delete the corresponding base(s) in s, summing over all possible deletion locations in all reads. Deletions with no weight are not included. Again, this weight has a probabilistic interpretation: it is the expected number of times we see the deletion in the reads, and again it may be fractional.

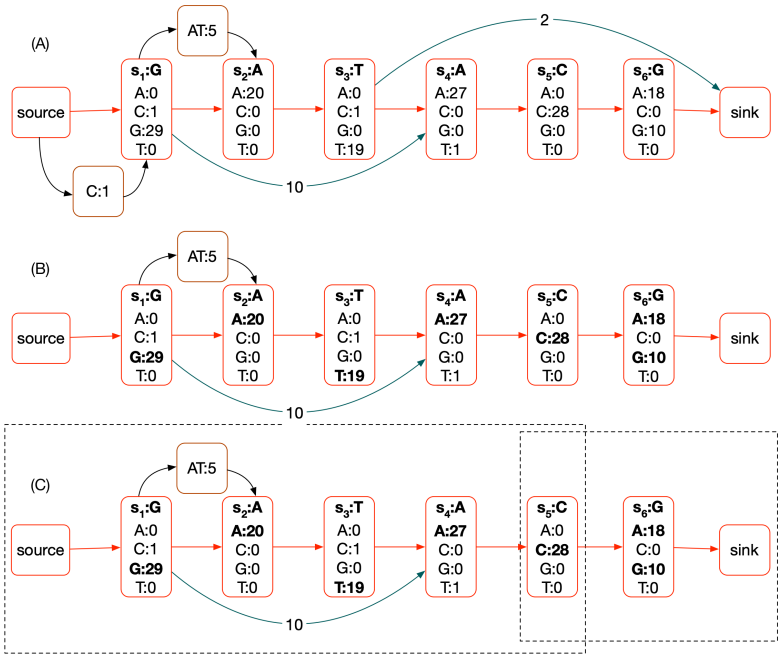

Figure C.9: A) An example POA, assuming approximately 30x read coverage. The backbone is shown in red. Each non-source/sink node has a vector of weights, one for each possible base. Deletion edges are shown in teal, they also each have a weight. Finally insertion nodes are shown in brown, each also has a weight. (B) A pruned POA, removing deletions and insertions that have less than a threshold weight and highlighting plausible bases in bold. There are six plausible nucleotide sequences represented by paths through the POA and selections of plausible base labels: G;AT;A;T;A;C:A, G;AT;A;T;A;C:G, G;A;T;A;C:A, G;A;T;A;C:G, G;A;C:A, G;A;C:G. To avoid the combinatorial explosion of such enumeration we identify subgraphs (C) and locally enumerate the possible subsequences in these regions independently (dotted rectangles identify subgraphs selected). In each subgraph there is a source and sink node that does not overlap any proposed edit.

We represent insertions as nodes labelled with an insertion sequence. Each insertion node has a single incoming edge from a backbone node, and a single outgoing edge to the next backbone node in the backbone sequence. Each insertion is labeled with a weight equal to the sum of probabilities of events that insert the given insertion sequence between the corresponding bases in s. The resulting POA is a restricted form of a weighted, directed acyclic graph (Fig. C.9(A) shows an example).

Frequently either an insertion or deletion can be made between different suc-

cessive bases in s resulting in the same edited sequence. To ensure that such equivalent events are not represented multiple times in the POA, and to ensure we sum their weights correctly, we 'left shift' indels to their maximum extent. When shifting an indel results in multiple equivalent deletion edges or insertions we remove the duplicate elements, updating the weight of the residual element to include the sum of the weights of the removed elements. For example, the insertion of 'AT' in Fig. C.9 is shifted left to its maximal extent, and could include the merger of an equivalent 'AT' insertion starting two backbone nodes to the right.

Local Haplotype Proposal After constructing the POA we use it to sample alternative assemblies. We first prune the POA to mark indels and base substitutions with weight below a threshold, which are generally the result of sequencing errors (Fig. C.9(B)). Currently this threshold (`polish.candidateVariantWeight`=0.18, established empirically) is normalized as a fraction of the estimated coverage at the site, which is calculated in a running window around each node in the backbone of 100 bases. Consequently if fewer than 18% of the reads are expected to include the change then the edit is pruned from consideration.

To further avoid a combinatorial explosion we sample alternative assemblies locally. We identify subgraphs of s containing indels and substitutions to s then in each subgraph, defined by a start and end backbone vertex, we enumerate all possible paths between the start and end vertex and all plausible base substitutions from the backbone sequence. The rationale for heuristically doing this locally is that two subgraphs sep-

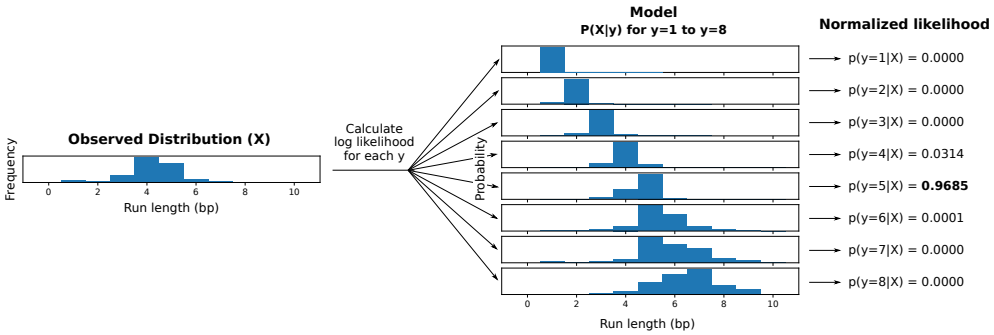

Figure C.10: **Visual representation of run length inference.** This diagram shows how a consensus run length is inferred for a set of aligned lengths (X) that pertain to a single position. The lengths are factored and then iterated over, and log likelihood is calculated for every possible true length up to a predefined limit. Note that in this example, the most frequent observation (4bp) is not the most likely true length (5bp) given the model.

arated by one or more *anchor* backbone sites with no plausible edits are conditionally independent of each other given the corresponding interstitial anchoring substring of s and the substrings of the reads aligning to it. Currently, any backbone site more than `polish.columnAnchorTrim`=5 nodes (equivalent to bases) in the backbone from a node overlapping a plausible edit (either substitution or indel) is considered an anchor. This heuristic allows for some exploration of alignment uncertainty around a potential edit. Given the set of anchors computation proceeds by identifying successive pairs of anchors separated by subgraphs containing the potential edits, with the two anchors considered the source and sink vertex.

A Simple Bayesian Model for Run-length Decoding Run-length encoding allows for separate modelling of length and nucleotide error profiles. In particular, length predictions are notoriously error prone in nanopore basecalling. Since

homopolymers produce continuous signals, and DNA translocates at a variable rate through the pore, the basecaller often fails to infer the true number of bases given a single sample. For this reason, a Bayesian model is used for error correction in the length domain, given a distribution of repeated samples at a locus.

To model the error profile, a suitable reference sequence is selected as the truth set. Reads and reference are run-length encoded and aligned by their nucleotides. The alignment is used to generate a mapping of observed lengths to their true length (y, x) where $y = true$ and $x = observed$ for each position in the alignment. Observations from alignment are tracked using a matrix of predefined size ($y_{max} = 50, x_{max} = 50$) in which each coordinate contains the corresponding count for (y, x). Finally the matrix is normalized along one axis to generate a probability distribution of $P(X|y_j)$ for j in $[1, y_{max}]$. This process is performed for each of the 4 bases.

With enough observations, the model can be used to find the most probable true run length given a vector of observed lengths X. This is done using a simple log likelihood calculation over the observations x_i for all possible true lengths y_j in Y, assuming the length observations to be independent and identically distributed. The length y_j corresponding to the greatest likelihood $P(X|y_j, Base)$ is chosen as the consensus length for each alignment position (Fig. C.10).

Training

To generate a model, we ran MarginPolish with reads from a specific basecaller version aligned to a reference (GRCh38) and specified the `--outputRepeatCounts` flag.

This option produces a TSV for each chunk describing all the observed repeat counts aligned to each backbone node in the POA. These files are consumed by a script in the `https://github.com/rlorigro/runlength_analysis` repository, which generates a RLE consensus sequence, aligns to the reference, and performs the described process to produce the model.

The `allParams.np.human.guppy-ff-235.json` model used for most of the analysis was generated from HG00733 reads basecalled with Guppy Flipflop v2.3.5 aligned to GRCh38, with chromosomes 1, 2, 3, 4, 5, 6, and 12 selected. The model `allParams.np.human.guppy-ff-233.json` was generated from Guppy Flipflop v2.3.3 data and chromosomes 1-10 were used. This model was also used for the CHM13 analysis, as the run-length error profile is very similar between v2.3.3 and v2.3.1 (v2.3.5 has a drastically different error profile, as is shown below in Fig. C.13).

Parallelization and Computational Considerations

To parallelize MarginPolish we break the assembly up into chunks of size `polish.chunkSize`=1000 bases, with an overlap of `polish.chunkBoundary`=50 bases. We then run the MarginPolish algorithm on each chunk independently and in parallel, stitching together the resulting chunks after finding an optimal pairwise alignment (using the default hmm described earlier) of the overlaps that we use to remove the duplication. We can further parallelize the algorithm across machines or processes using a provided Toil script CITE:PMID: 28398314.

Memory usage scales with thread count, read depth, and chunk size. For

this reason, we downsample reads in a chunk to `polish.maxDepth`=50× coverage by counting total nucleotides in the chunk N_c and discarding reads with likelihood $1 - (\texttt{chunkSize} + 2 * \texttt{chunkBoundary}) * \texttt{maxDepth}/N_c$. With these parameters, we find that 2GB of memory per thread is sufficient to run MarginPolish on genome-scale assemblies. Across 13 whole-genome runs, we averaged roughly 350 CPU hours per gigabase of assembled sequence.

HELEN: Homopolymer Encoded Long-read Error-corrector for Nanopore

HELEN is a deep neural network based haploid consensus sequence polisher. HELEN employs a multi-task recurrent neural network (RNN) [93] that takes the weights of the partial order alignment (POA) graph of MarginPolish to predict a base and a run-length for each genomic position. MarginPolish constructs the POA graph by performing multiple possible alignments of a single read that makes the weights associative to the correct underlying base and a run-length. The RNN employed in HELEN takes advantage of the transitive relationship of the genomic sequence and associative coupling of the POA weights to the correct base and run-length to produce a consensus sequence with higher accuracy.

The error-correction with HELEN is done in three steps. First, we generate tensor-like images of genomic segments with MarginPolish that encodes POA graph weights for each genomic position. Then we use a trained RNN model to produce

predicted bases and run-lengths for each of the generated images. Finally, we stitch the chunked sequences to get a contiguous polished sequence.

Image Generation

MarginPolish produces an image-like summary of the final POA state for use by HELEN. At a high level, the image summarizes the weighted alignment likelihoods of all reads divided into nucleotide, orientation, and run-length.

The positions of the POA nodes are recorded using three coordinates: the position in the backbone sequence of the POA, the position in the insert sequences between backbone nodes, and the index of the run-length block. All backbone positions have an insert coordinate of 0. Each backbone and insert coordinate includes one or more run-length coordinate.

When encoding a run-length, we divide all read observations into blocks from 0 to 10 inclusive (this length is configurable). For cases where no observations exceed the maximum run-length, a single run-length image can describe the POA node. When an observed run-length exceeds the length of the block, the run-length is encoded as that block's maximum (10), and the remaining run-length is encoded in successive blocks. For a run-length that terminates in a block, its weight is contributed to the run-length 0 column in all successive blocks. This means that the records for all run-length blocks of a given backbone and insert position have the same total weight. As an example, consider three read positions aligned to a node with run-lengths of 8, 10, and 12. These require two run-length blocks to describe: the first block includes one 8 and two 10s,

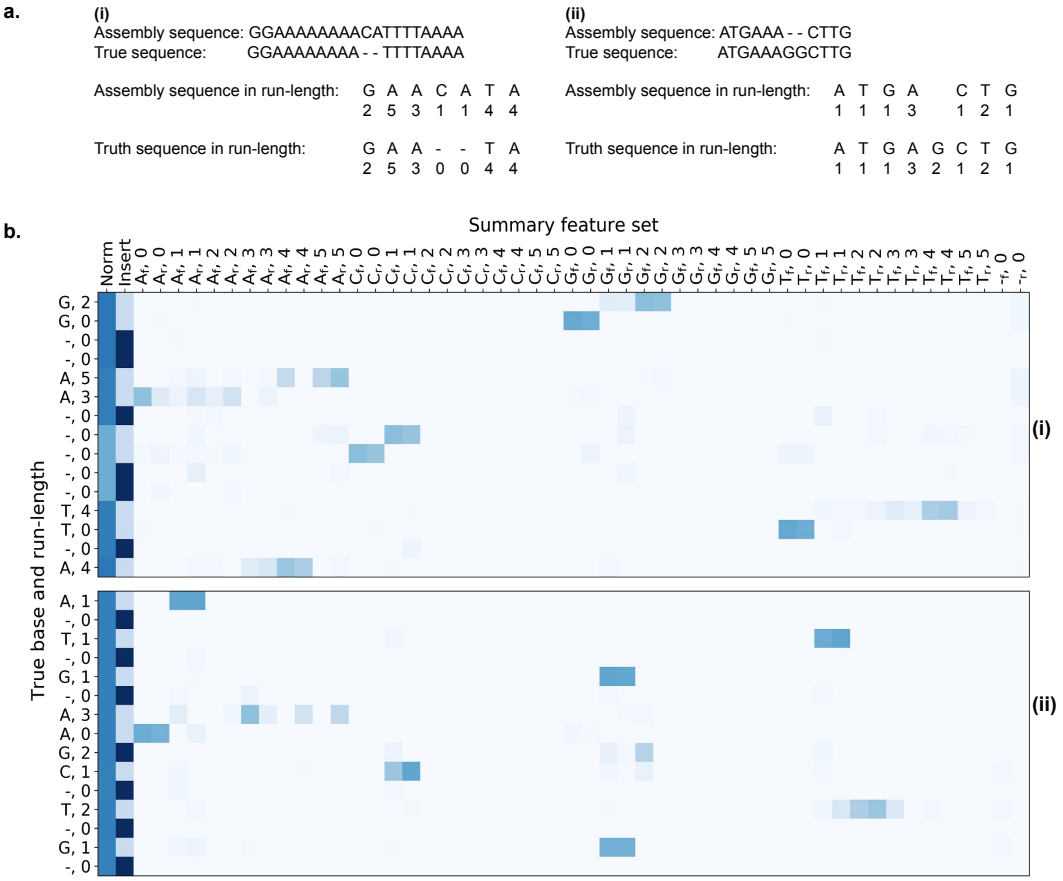

Figure C.11: **MarginPolish Images** A graphical representation of images from two labeled regions selected to demonstrate: the encoding of a single POA node into two run-length blocks (i), a true deletion (i), and a true insert (ii). The y-axis shows truth labels for nucleotides and run-lengths, the x-axis describes features in the images, and colors show associated weights.

and the second includes two 0s and one 2.

The information described at each position (backbone, insert, and run-length) is encoded in 92 features: each nucleotide {A, C, T, G} and run-length {0, 1, .., 10}, plus a gap weight (for deletions in read alignments). The weights for each of these 45 observations are separated into forward and reverse strand for a total of 90 features. The weights for each of these features are normalized over the total weight for the record and accompanied by an additional data point describing the total weight of the record. This normalization column for the record is an approximation of the read depth aligned to that node. Insert nodes are annotated with a binary feature (for a final total of 92); weights for an insert node's alignments are normalized over total weight at the backbone node it is rooted at (not the weight of the insert node itself) and gap alignment weights are not applied to them.

Labeling nodes for training requires a truth sequence aligned to the assembly reference. This provides a genome-scale location for the true sequence and allows the its length to help in the resolution of segmental duplications or repetitive regions. When a region of the assembly is analyzed with MarginPolish, the truth sequences aligned to that region are extracted. If there is not a single truth sequence which approximately matches the length of the consensus for this region, we treat it as an uncertain region and no training images are produced. Having identified a suitable truth sequence, it is aligned to the final consensus sequence in non-run-length space with Smith-Waterman. Both sequences and the alignment are then run-length encoded, and true labels are matched with locations in the images. All data between the first and last matched

nodes are used in the final training images (leading and trailing inserts or deletes are discarded). For our training, we aligned the truth sequences with `minimap2` using the `asm20` preset and filtered the alignments to include only primary and supplementary alignments (no secondary alignments).

Fig. C.11 shows a graphical representation of the images. On the y-axis we display true nucleotide labels (with the dash representing no alignment / gap) and true run-length. On the x-axis the features used as input to HELEN are displayed: first the normalization column (the total weight at the backbone position), second the insert column (the binary feature encoding whether the image is for a backbone or insert node), forty-eight columns describing the weights associated with read observations (stratified by nucleotide, run-length, strand), and two columns describing weights for gaps in read alignments (stratified by strand). In this example, we have reduced the maximum run-length per block from 10 to 5 for demonstrative purposes.

We selected these two images to highlight three features of the model: the way multiple run-length blocks are used to encode observations for a single node, and the relevant features around a true gap and a true insert that enable HELEN to correct these errors.

To illustrate multiple run length blocks, we highlight two locations on on image (i). The first are the nodes labeled (A,5) and (A,3). This is the labeling for a true (A,8) sequence separated into two blocks. See that the bulk of the weight is on the (A,5) features on the first block, with most of that distributed across the (A,1-3) features on the second. Second, observe the nodes on (i) labeled (T,4) and (T,0). Here we show

the true labeling of a (T,4) sequence where there are some read observations extending into a second run-length block.

To show a features of a true gap, note on (i) the non-insert nodes labeled (-,0). We know that MarginPolish predicted a single cytosine nucleotide (as it is a backbone node and the (C,1) nodes have the bulk of the weight. Here, HELEN is able to use the low overall weight (the lighter region in the normalization column) at this location as evidence of fewer supporting read alignments and can correct the call.

The position labeled (G,2) on (ii) details a true insertion. It is not detected by MarginPolish (as all insert nodes are not included in the final consensus sequence). Read support is present for the insert, less than the backbone nodes in this image but more than the other insert nodes. HELEN can identify this sequence and correct it.

Finally, we note that the length of the run length blocks results in streaks at multiples of this length (10) for long homopolymers. The root of this effect lies in the basecaller producing similar prediction distributions for these cases (ie, the run length predictions made by the basecaller for a true run length of 25 are similar to the run length predictions made for a true run length of 35, see Fig. 3.4b Guppy 2.3.3). This gives the model little information to differentiate upon, and the issue is exacerbated by the low occurrence of long run lengths in the training data. Because the model divides run length observations into chunks of size 10, it tends to call the first chunks correctly (having length 10) but has very low signal for the last chunk and most often predicts 0.

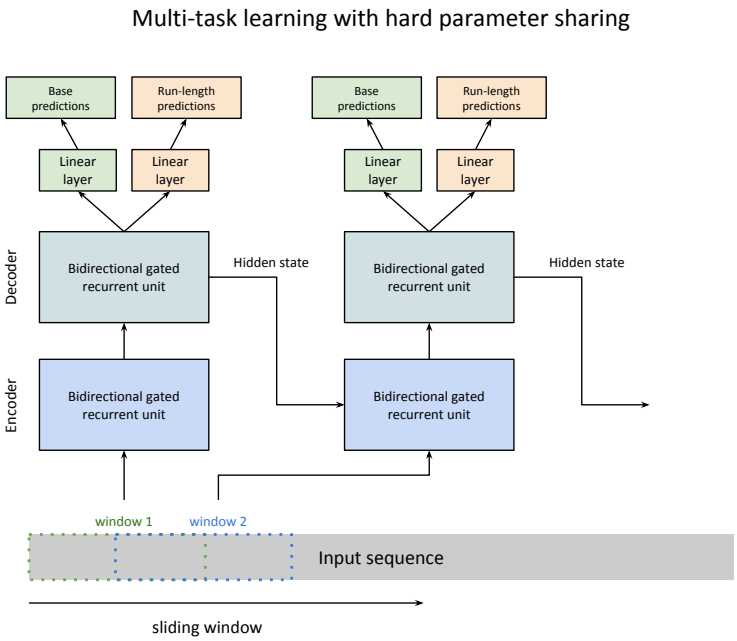

Figure C.12: The sequence-to-sequence model implemented in Helen.

The model

We use a sequence transduction model for consensus polishing. The model structure consists of two single-layer gated recurrent neural units (GRU) for encoding and decoding on top of two linear transformation layers. The two linear transformation layers independently predict a base and a run-length for each position in the input sequence. Each unit of the GRU can be described using the four functions it calculates:

$$
\begin{aligned}
r_t &= Sigmoid(W_{ir}x_t + W_{hr}h_{(t-1)}) \\
u_t &= Sigmoid(W_{iu}x_t + W_{hu}h_{(t-1)}) \\
n_t &= tanh(W_{in}x_t + r_t * (W_{hn}h_{(t-1)})) \\
h_t &= (1 - u_t) * n_t + u_t * h_{(t-1)}
\end{aligned}
\tag{C.1}
$$

For each genomic position t, we calculate the current state h_t from the new state n_t and the update value u_t applied to the output state of previous genomic position $h_{(t-1)}$. The update function u_t decides how much past information to propagate to the next genomic position. It multiplies the input x_t with the weight vector W_{iu} and multiplies the hidden state of the previous genomic position $h_{(t-1)}$. The weight vectors decide how much from the previous state to propagate to the next state. The reset function r_t decides how much information to dissolve from the previous state. Using a different weight vector, the r_t function decides how much information to dissolve from the past. The new memory state n_t is calculated by multiplying the input x_t with the weight vector W_{in} and applying a Hadamard multiplication $*$ between the reset function value and a weighted state of the previous hidden state $h_{(t-1)}$. The new state captures the associative relationship between the input weights and true prediction. In this setup, we can see that r_t and u_t can decide to hold memory from distant locations while n_t captures the associative nature of the weights to the prediction, helping the model to decide how to propagate genomic information in the sequence. The output of each genomic position h_t can be then fed to the next genomic position as a reference to the previously decoded genomic position. The final two layers apply linear transformation functions:

$$\begin{aligned} B_t &= h_t * W^T \\ R_t &= h_t * W^T \end{aligned} \tag{C.2}$$

The two linear transformation functions independently calculate a base prediction B_t and a run-length prediction R_t from the hidden state output of that genomic position

h_t. The model operates in hard parameter sharing mode where the model learns to perform two tasks in equation C.2 using the same set of underlying parameters from equation C.1. The ability of the model to reduce the error rate of the assemblies from multiple samples with multiple assemblers shows the generalizability and robustness we achieve with this method.

Sliding window mechanism

One of the challenges of this setup is the sequence length. From the functions of recurrent units in equation C.1, we see that each state is updated based on the previous state and associated weight. Due to the noisy nature of the data, if the sequence length is too long, the back-propagation becomes difficult over noisy regions. On the other hand, a small sequence length would make the program very slow. We balance the run-time and accuracy by using a sliding window approach.

During the sliding-window, we chunk the sequence of thousand bases to multiple overlapping windows of length 100. Starting from the leftmost window, we perform prediction on sequence pileups of the window and transmit the hidden state of the current window to the next window and slide the window by 50 bases to the right. For each window, we collect all the predicted values and add it to a global sequence inference counter that can keep track of predicted probabilities of base and run-length at each position. Lastly, we aggregate the probabilities from the global inference counter to generate a sequence. This setup allows us to utilize the minibatch feature of the popular neural network libraries allowing inference on a batch of inputs instead of performing

inference one at a time.

Training the model

HELEN is trained with a gradient descent method. We use Adaptive Moment Estimation (Adam) method to compute gradients for each of the parameters in the model based on a target loss function. Adam uses both decaying squared gradients and the decaying average of gradients, making it suitable to use with recurrent neural networks[93]. Adam performs gradient optimization by adapting the parameters to set in a way that minimizes the value of the loss function.

We perform optimization through back-propagation per window of the input sequence. From equation C.2, we see that we get two vectors $B = [B_1, B_2, B_3...B_n]$ and $R = [R_1, R_2, R_3...R_n]$ containing base and run-length predictions for each window of size n. From the labeled data we get two more such vectors $T_B = [T_{B1}, T_{B2}, T_{B3}, ...T_{Bn}]$ and $T_R = [T_{R1}, T_{R2}, T_{R3}, ...T_{Rn}]$ containing the true base and true rle values of each position in the window. From these loss function the loss L is calculated:

$$
\begin{aligned}
L_B(B, T_B) &= -B[T_B] + \log\left(\textstyle\sum_j \exp(B[j])\right) \\
L_R(R, T_R) &= weight[T_R]\left(-R[T_R] + \log\left(\textstyle\sum_j \exp(R[j])\right)\right) \\
L &= L_B + L_R
\end{aligned}
\tag{C.3}
$$

In equation C.3, L_B calculates the base prediction loss and L_R calculates the rle prediction loss. The rle class distribution is heavily biased toward lower run-length values, so, we apply class-wise weights depending on the observation of per class to make the

learning process balanced between classes. The optimizer then updates the parameters or weights W of the model from equation C.1 and equation C.2 in a way that minimizes the value of the loss function. We can see that the loss function is a summation of the two independent loss functions but the underlying weights from the recurrent neural network belongs to the same set of elements in the model. In this setting, the model optimizes to learn both task simultaneously by updating the same set of weights.

Sequence stitching

To parallelize the polishing pipeline, MarginPolish chunks the genome into smaller segments while generating images. Each image segment encodes a thousand nucleotide bases, and two adjacent chunks have 50 nucleotide bases overlap between them. During the inference step, we save all run-length and base predictions of the images, including their start and end genomic positions.

For stitching, we load all the image predictions and sort them based on the genomic start position of the image chunk and stitch them in parallel processes. For example, if there are n predictions from n images of a contig and we have t available threads, we divide n prediction chunks into t buckets each containing approximately n/t predicted sequences. Then we start t processes in parallel where each process stitches all the sequences assigned to it and returns a longer sequence. For stitching two adjacent sequences, we take the overlapping sequences between the two sequence and perform a pairwise Smith-Waterman alignment. From the alignment, we pick an anchor position where both sequences agree the most and create one sequence. After all

the processes finish stitching the buckets, we get t longer sequences generated by each process. Finally, we iteratively stitch the t sequences using the same process and get one contiguous sequence for the contig.

Generating trained models

In supplementary tables D.16, D.19 and D.18 we report several models for HELEN. The models are trained on different sets of data with varying Guppy base-caller versions. We discuss three trained models: r941_flip235_v001.pkl, r941_flip233_v001.pkl, and r941_flip231_v001.pkl to use with HELEN for different versions of the ONT Guppy base-callers. Due to the difference in the error profile of different versions of the Guppy base-caller, we trained three different models.

Table C.1: Description of trained models for HELEN.

Model Name	Base caller version	Training sample	Training region	Testing region
r941_flip235_v001.pkl	Guppy 2.3.5	HG002	Chr1-19, Chr21-22	Chr20
r941_flip233_v001.pkl	Guppy 2.3.3	HG002	Chr1-19, Chr21-22	Chr20
r941_flip231_v001.pkl	Guppy 2.3.1	CHM13	Chr1-6	Chr20

The `r941_flip235_v001.pkl` is trained on HG002 base called with Guppy 2.3.5. The model is trained on the high confidence regions of all autosomes and tested on Chr20. The training script trained the model for 80 hours on 10 epochs, which generated 10 trained models. We picked the model that has the best performance on

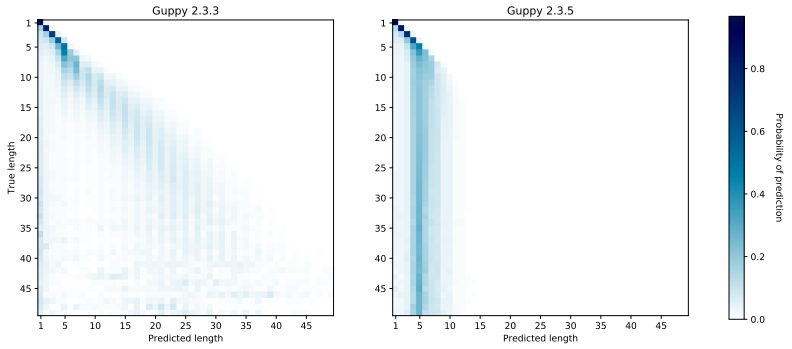

Figure C.13: Run-length confusion in different versions of Guppy base caller

Chr20 as the final model.

The CHM13 data from T2T consortium [84] were base called with Guppy 2.3.1. The error profile of Guppy 2.3.1 is significantly different than Guppy 2.3.5. Figure C.13 shows the difference in underlying error profile of HG00733 sample for two different versions of Guppy. We trained `r941_flip233_v001.pkl` Model on HG002 Guppy 2.3.3 data. Although the error profile of Guppy 2.3.1 and Guppy 2.3.3 are similar, the reported base qualities are different. So, we trained another model `r941_flip231_v001.pkl` on Chr1-6 of CHM13 to see further improvement in the consensus quality of CHM13.

Implementation notes

We have implemented HELEN using python and C++ programming language. We use PyTorch [141] deep neural network library for the model implementation. We also use the Striped-Smith Waterman algorithm implementation to use during stitching and Pybind11 [142] as a bridge between C++ and python methods. The image data is saved using HDF5 file format. The implementation is publicly available via GitHub

(https://github.com/kishwarshafin/helen).

Appendix D

Shasta appendix

The following contains relevant figures and tables from the supplement of the Shasta paper.

Supplementary Notes

Execution Parameters

Shasta

All Shasta runs used Shasta version 0.1.0 built from `https://github.com/chanzuckerberg/shasta`. Rather than using the distributed version of the release, the source code was rebuilt locally for best performance as recommended by Shasta documentation.

The Shasta executable was run with the following command:

```
shasta --memoryMode filesystem --memoryBacking 2M
```

Canu

Canu 1.8 from `https://github.com/marbl/canu` was run with the following command:

```
canu -p asm -d asm genomeSize=3.1g
    'corMhapOptions=--threshold 0.8 --num-hashes 512
                --ordered-sketch-size 1000 --ordered-kmer-size 14'
    'gridOptionsJobName=mom'
    'gridOptions=--time=240:00:00 --partition=norm'
    'stageDirectory=/lscratch/$SLURM_JOBID'
    'gridEngineStageOption=--gres=lscratch:100'
    'correctedErrorRate=0.105'
    -nanopore-raw input.fastq.gz
```

Wtdbg2

Wtdbg2 version 2.3 from `https://github.com/ruanjue/wtdbg2` was run with the following commands:

```
wtdbg2 -t 0 -x ont -L 10000 -g 3.3g
    -i reads1.fastq.gz -i reads2.fastq.gz -i reads3.fastq.gz
    -o wtdbg2-assembly
wtpoa-cns -t 31 -i wtdbg2-assembly.ctg.lay.gz -f
    -o wtdbg2-assembly.fa
```

Flye

Flye version 2.4.2 from `https://github.com/fenderglass/Flye` was run with the following command:

```
flye --nano-raw reads1.10kb.fastq.gz reads2.10kb.fastq.gz
        reads3.10kb.fastq.gz
    --genome-size 3.3g --out-dir flye  --threads 123
```

Racon

We used a home-grown script to manage running 4 iterations of Racon, v1.3.2. The code for the script can be found here `https://github.com/rlorigro/nanopore_assembly_and_polishing_assessment`, and was run with the following command:

```
python3 nanopore_assembly_and_polishing_assessment/polish.py
    --true_ref hg38.fa
    --contigs assembly.fasta
    --sequences reads.fasta
    --output_dir racon
    --n_passes 4
```

When run for the analysis to produce Supplemental Table D.22, the `n_passes` parameter was set to 1.

Medaka

Medaka version 0.6.0-alpha.3 (`https://github.com/nanoporetech/medaka`) was run with the following commands:

```
medaka consensus -i reads5.fasta -d assembly_racon4x.fasta
    -o medaka -t 64 -m r941_flip235
medaka stitch medaka/consensus_probs.hdf medaka/consensus.fasta
```

No changes in the arguments were used for the analysis that produced Supplemental Table D.22. This includes the GPU mode, which is configured during compilation.

Minialign

Minialign is bundled with Medaka, and was run with the following commands:

```
mini_align -i reads.fasta -r assembly.fasta
    -P -m -p medaka/calls_to_draft -t 60
```

Minimap2, Samtools

Minimap2 version 2.15-r908-dirty from `https://github.com/lh3/minimap2`. We used samtools 1.7 using htslib 1.7-2 for sorting and filtering. The following three commands were piped into each other:

```
minimap2 -ax map-ont -t 70 assembly.fasta reads.fasta
samtools sort -@ 70
```

```
samtools view -hb -F 0x104 > align.bam
```

MarginPolish

MarginPolish 1.0.0 was compiled from `https://github.com/UCSC-nanopore-cgl/MarginPolish` run with the following command:

```
marginPolish input.bam input.fa allParams.np.human.guppy-ff-235.json
    -f -o output_location -t 70
```

When run to produce Supplemental Table D.22, MarginPolish was used compiled from the commit `4c1da1e1b3efc739e9c48913416efac619d3d40c` on GitHub.

HELEN

HELEN version 0.1 from `https://github.com/kishwarshafin/helen` was run with the following commands:

```
python3 /home/ubuntu/software/helen/call_consensus.py
    -i images/ -b 1024 -w 16 -t 32 -m r941_flip235_v001.pkl -o out -g


python3 /home/ubuntu/software/helen/stitch.py
    -i out/helen_predictions_05312019_183902.hdf -o out/
    -p polished_assembly -t 32
```

HiRise

HiRise was run via a docker container, with access given by Dovetail Genomics. The HiRise version was v2.1.6, with the HiRise Helper version 2.1.10 and the HiRise Utils version v2.1.7-3-g98c1a1b. Default parameters were used.

Long Ranger

The 10X Long Ranger Align pipeline (v2.2) was used for any alignment of 10X reads to a reference. An example sequence of commands was:

```
longranger mkref assembly.fa
longranger align --id 10x-chm13-chrX-round1
    --reference refdata-assembly --fastqs fastq/
```

Pilon

An example Pilon command (using v1.23) is below:

```
java -Xmx200G -jar pilon-1.23.jar --bam align.bam --genome assembly.fa
    --threads 32 --output pilon-out
```

Trio-binning

For HG00733, the parental read sample accessions were obtained from 1000 genome database:

```
http://www.internationalgenome.org/data-portal/sample/HG00731
```

http://www.internationalgenome.org/data-portal/sample/HG00732

Briefly, k-mers were counted with `meryl`, subtracted to generate maternal/paternal sets, and any k-mers occurring less than 6 times for maternal k-mers and 5 times for paternal k-mers were not used. Binning did not use normalization by k-mer set size. This resulted in 35.2x maternal, 37.3x paternal, and 5.6x unclassified. Assembly did not use the unclassified reads and ran with the command:

```
canu -p asm -d <mom/dad> 'genomeSize=3.1g'
    'corMhapOptions=--threshold 0.8 --num-hashes 512
                --ordered-sketch-size 1000 --ordered-kmer-size 14'
    'corMinCoverage=0'
```

Each haplotype assembly required approximately 100k CPU hours (4-5 days). A subsequent run using Canu 1.8 and automated binning with the command:

```
canu -p asm -d asm 'genomeSize=3.1g'
    'corMhapOptions=--threshold 0.8 --num-hashes 512
                --ordered-sketch-size 1000 --ordered-kmer-size 14'
    'gridOptionsJobName=733_trio'
    'corMinCoverage=0'
    -haplotypeMOM hg0732/*fastq.gz
    -haplotypeDAD hg0731/*.fastq.gz
```

resulted in a similar classification split (35.1x dad, 36.7x mom, 5.6x unknown) and

assembly (manual: dad=16.6 NG50, mom=18.1 NG50; automated: dad=14.1 NG50, mom=19.9 NG50).

For HG0002, Illumina data for the parents was downloaded from the GIAB ftp site:

```
ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG003_NA24149_father
    /NIST_HiSeq_HG003_Homogeneity-12389378/HG003_HiSeq300x_fastq/
ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG004_NA24143_mother
    /NIST_HiSeq_HG004_Homogeneity-14572558/HG004_HiSeq300x_fastq/
```

K-mers were counted as before, subtracted, and filtered to exclude k-mers occuring less than 25 times in the maternal or paternal set. The classification resulted in 24x maternal, 23x paternal, and 3.5x unknown. Only classified reads were used for assembly with the command:

```
canu -p asm -d <mom/dad> 'genomeSize=3.1g'
    'corMhapOptions=--threshold 0.8 --num-hashes 512
                --ordered-sketch-size 1000 --ordered-kmer-size 14'
    'corMinCoverage=0'
```

Each haplotype assembly required approximately 100k cpu hours (4-5 days).

QUAST

When using QUAST to evaluate assembly statistics and run BUSCO, we used the following command below. `--large` indicates that the genome is large, `--fragmented` indicates the reference genome may be fragmented, `--min-identity 80` indicates that

alignments with identity less than 80% will be filtered, `--conserved-genes-finding` indicates that BUSCO will be run to find universal single-copy orthologs, and `eukaryote` indicates that the genome is from a eukaryote.

```
quast-lg.py --threads 12 -r truth_assembly.fa -o quast-out --large
    --min-identity 80 --fragmented --conserved-genes-finding --eukaryote
    assembly.fa
```

Benchmarking assemblies using Pomoxis

The truth assembly files and the reported error-rates are described in Online methods.

To benchmark the assemblies, we used `assess_assembly` pull 37 from Pomoxis (`https://github.com/nanoporetech/pomoxis/pull/37`). This tool is developed and suggested by the research group of Oxford Nanopore Technology. We added the functionality to ignore large insertions and deletions. The installation instruction of Pomoxis can be found on the github page `https://github.com/nanoporetech/pomoxis`. The parameters we used are:

- -i: The input assembly (fasta).
- -r: The reference fasta file. (The truth assembly)
- -b: Bed file containing reference regions to assess.
- -p: Prefix of the output file names.

- -c: Chunk size. Input reads/contigs will be broken into chunks prior to alignment.
- -t: Number of threads to use.
- -T: Trim consensus to primary alignments of truth to assembly.
- -l: Ignore insertions and deletions longer than this value, 0 means include everything. (default 0)

We compared the HG002 samples, we gathered the truth assembly file, a bed file describing the confident regions, and a shasta assembly and ran the following command.

```
assess_assembly -i hg002_shasta_assembly.fa -r hg002_truth_assembly.fa
    -b hg002_confident.bed -p hg002_shasta_assessment
    -c 1000 -l 50 -t 32 -T
```

In this setup, the `assess_assembly` module computes the error rate of the input `hg002_shasta_assembly.fa` that aligns to the high-confidence region defined in the `hg002_confident.bed` of `hg002_truth_assembly.fa` assembly. Also, the `-T` parameter limits the assessment to regions where there is an alignment between the truth and the input assembly.

For HG00733 sample, we used the high-quality phased PacBio assembly. We got `hg00733_truth_assembly.fa` and the `hg00733_shasta_assembly.fa` and ran the following command for assessment.

```
assess_assembly -i hg00733_shasta_assembly.fa
```

```
    -r hg00733_truth_assembly.fa -p hg00733_shasta_assessment
    -t 32 -c 1000 -l 50 -T
```

As the truth assembly of HG00733 does not define any high-confidence region, we do a whole genome comparison where there is an alignment between the truth and the input assembly enforced by the `-T` parameter. For CHM13 and all other assemblies, we used the same command as HG00733. The output of this program reports different error rates described in the online methods section.

Extracting common assembly regions

To create a bed file describing the regions where all the assemblers have an assembly, we used `mini_align` available `https://github.com/nanoporetech/pomoxis/`, and `bedtools` which can be found in `https://bedtools.readthedocs.io/en/latest/`.

We first align the assembly to the truth assembly using `mini_align`.

```
mini_align -P -m -c 100000 -r truth_assembly.fa
    -i assembler_assembly.fa -t 64 -p assembler_2_truth
```

Then we extract the regions where the assemblers have an assembly:

```
bedtools bamtobed -i assembler_2_truth.bam > assembler.bed
```

Finally we do an intersection of all the bed files that we get from each assemblers. For HG002, we also included the high confidence region bed file.

```
multiIntersectBed -i <list_of_bed> | awk '$4 == <number_of_beds>'
```

```
    > common_regions.bed
sort -k1,1 -k2,2n common_regions_between_assemblers_hg002.bed
    > common_regions.sort.bed
bedtools merge -i common_regions.sort.bed
    > common_regions_between_assemblers.bed
```

Extracting chrX from assemblies

To analyze subsets of the CHM13 assemblies which correspond to regions in chrX, we used the following steps to extract contigs. Briefly, we align the assembly to GRCh38, identify any assembly contig which had a primary or supplementary alignment to chrX, and extract these segments.

```
minimap2 -ax asm20 -t 32 GRCh38.fa assembly.fa | samtools view -hb
    > unsorted.bam
samtools sort -@ 32 unsorted.bam | samtools view -hb
    > assembly.bam
samtools index -@ 32 assembly.bam
samtools view -F 0x104 assembly.bam chrX | awk '{print $1}' | sort | uniq
    > segments.txt
extract_fasta_segments.py -i assembly.fa -s segments.txt
    -o assembly.hg38_chrX.fa
```

The script `extract_fasta_segments.py` can be found at `https://github.com/tpesout/genomics_scripts`.

Supplementary Results

Shasta: assembling a human genome from nanopore reads in under 6 hours

Table D.1: QUAST assembly metrics of three samples on four assemblers before polishing, compared against GRCh38 with no alternate contigs.

Sample	Metric	Shasta	Wtdbg2	Flye	Canu
HG00733	# contigs	2,150	5,086	1,852	778
	Total length	2,783,599,890	2,792,376,827	2,816,034,584	2,900,719,051
	N50	24,429,871	18,763,119	28,763,002	44,759,083
	NG50	21,088,309	15,338,021	25,227,330	40,627,903
	# disagreements	814	3,985	6,555	4,570
	Genome fraction (%)	94.982	92.938	95.763	96.404
	Duplication ratio	0.995	1.005	0.986	1.014
	# mismatches per 100 kbp	156.21	248.78	506.12	231.24
	# indels per 100 kbp	453.97	664.90	1,480.91	677.26
	Total aligned length	2,775,307,347	2,742,343,142	2,769,440,009	2,858,769,830
	NA50	16,052,981	9,106,500	18,577,806	21,157,324
	NGA50	12,765,264	7,787,949	16,267,214	19,945,150
HG002	# contigs	1,847	5,310	1,627	767
	Total length	2,801,200,983	2,793,889,694	2,819,241,152	2,901,099,163
	N50	23,346,484	15,380,722	31,253,170	33,064,788
	NG50	20,205,529	13,750,884	25,917,293	32,340,595
	# disagreements	901	3,572	5,881	3,882
	Genome fraction (%)	95.622	93.136	96.228	96.959
	Duplication ratio	0.995	1.004	0.981	1.009
	# mismatches per 100 kbp	167.75	261.72	549.10	231.39
	# indels per 100 kbp	520.33	796.71	1,650.63	792.45
	Total aligned length	2,792,458,737	2,743,401,414	2,768,347,339	2,863,787,213
	NA50	16,068,951	8,564,600	18,803,788	21,330,391
	NGA50	14,189,972	7,361,363	16,079,132	18,175,258
CHM13	# contigs	1,236	6,428	1,269	558
	Total length	2,809,087,051	2,836,802,421	2,857,931,691	2,919,690,848
	N50	46,037,322	15,522,332	36,829,446	80,507,947
	NG50	41,091,906	14,039,241	35,319,460	79,504,166
	# disagreements	1,051	4,202	5,452	4,768
	Genome fraction (%)	95.307	93.124	96.022	96.553
	Duplication ratio	1.000	1.017	0.997	1.014
	# mismatches per 100 kbp	155.15	256.17	443.85	226.04
	# indels per 100 kbp	358.45	535.46	1,023.79	484.46
	Total aligned length	2,798,043,587	2,780,449,715	2,807,157,420	2,864,418,837
	NA50	23,475,255	6,786,237	18,991,999	25,611,947
	NGA50	18,990,051	5,892,796	17,032,972	23,819,455

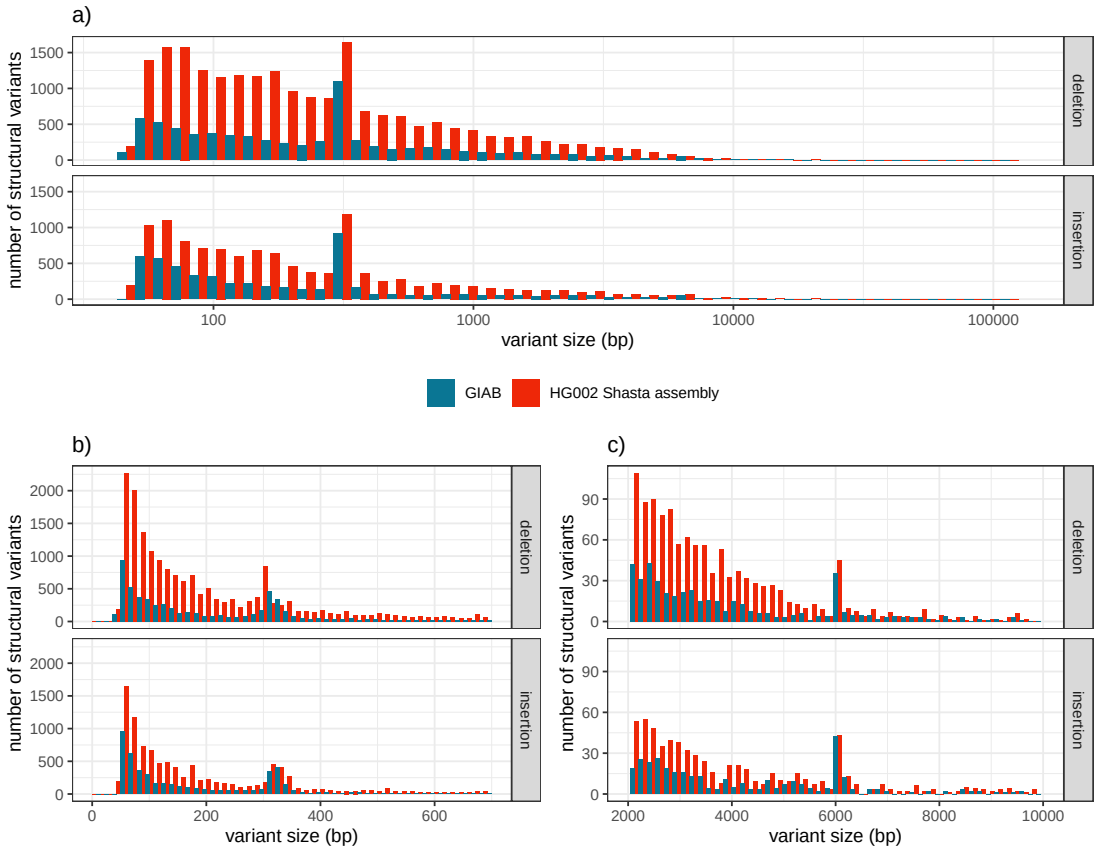

Figure D.1: Size distribution of structural variants (¿50 bp) extracted from the Shasta assembly graph for HG002 and the structural variants in the Genome In A Bottle (GIAB) catalog for the same sample. a) Full size distribution for deletions (top) and insertion (bottom), in log-scale. b) and c) zoom in the two peaks caused by Alu (˜300 bp) and L1 (˜6 Kbp) insertion polymorphisms.

Table D.2: QUAST disagreement count for four assemblers on different regions of the genome for four samples. We report disagreements that happen in all chromosomes of GRCh38, then incrementally exclude centromeric regions, segmental duplication regions (Seg Dups), and all other regions enriched for SVs (chrY, acrocentric chromosome arms, and QH-regions)

Sample	Assembler	Disagreements in GRCh38 autosomes and chrX, chrY	Disagreements outside centromeres	Disagreements outside centromeres and seg dups	Disagreements outside centromeres, seg dups, chrY, acrocentric chr arms, and QH-regions
HG002	Shasta	901	755	284	121
	Flye	5881	1226	513	117
	Canu	3882	2347	689	216
	Wtdbg2	3572	1213	484	148
HG00733	Shasta	814	662	256	110
	Flye	6555	1261	604	134
	Canu	4570	2791	755	224
	Wtdbg2	3985	1166	474	135
CHM13	Shasta	1051	795	333	129
	Flye	5452	1228	448	107
	Canu	4768	2764	864	164
	Wtdbg2	4202	1519	592	249

Table D.3: Disagreement count in the intersection of the assemblies for each sample (see Online Methods). Total Disagreements describes all disagreements found in 100bp windows before taking the intersection; note that these counts are very close to those reported by QUAST. Consensus Disagreements describes disagreements in the intersection of the four assemblies. Genome fraction describes total coverage over GRCh38 for the consensus sequence.

Sample	Assembler	Total Disagreements	Consensus Disagreements	Genome Fraction
HG002	Shasta	863	179	87.16%
	Flye	5823	178	87.16%
	Canu	3779	328	87.16%
	Wtdbg2	3509	215	87.16%
HG00733	Shasta	792	161	87.43%
	Flye	6546	178	87.43%
	Canu	4524	383	87.43%
	Wtdbg2	3975	205	87.43%
CHM13	Shasta	1033	242	87.53%
	Flye	5446	217	87.53%
	Canu	4682	712	87.53%
	Wtdbg2	4190	404	87.53%

Table D.4: Disagreement count and fraction of genome covered on chromosome X for four assemblers on CHM13 assemblies with no polishing, compared to the chromosome X assembly from the Telomere-to-Telomere Consortium. These numbers were obtained via running QUAST.

Assembler	Disagreements	Genome Fraction
Shasta	5	97.73%
Wtdbg2	87	94.17%
Flye	18	98.41%
Canu	9	98.16%

Table D.5: BAC analysis on selected dataset. BACs were selected (31 of CHM13 and 16 of HG00733) for falling within unique regions of the genome, specifically >10 Kb away from the closest segmental duplication. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus in the assembly. *Attempted* refers to the number of BACs which have an alignment for >5 Kb of sequence with >90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs.

Sample	Assembler	BAC counts				Median Quality		Mean Quality	
		Total	Attempted	Closed	Closed of attempted %	Identity %	QV	Identity %	QV
CHM13	Canu	31	31	30	96.77	99.40	22.18	99.34	21.84
	Flye	31	31	31	100.00	97.58	16.17	97.65	16.28
	Shasta	31	31	31	100.00	99.55	23.51	99.51	23.07
	Wtdbg2	31	29	28	96.55	99.46	22.71	99.39	22.15
HG00733	Canu	16	16	15	93.75	98.74	18.98	98.61	18.56
	Flye	16	16	16	100	97.99	16.97	98.01	17.02
	Shasta	16	16	16	100	98.84	19.38	98.79	19.20
	Wtdbg2	16	16	16	100	98.81	19.26	98.79	19.20

Table D.6: BAC analysis on full dataset, 341 on CHM13 and 179 on HG00733. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus. *Attempted* refers to the number of BACs which have an alignment for ¿5Kb of sequence with ¿90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs.

Sample	Assembler Polisher	BAC counts				Median Quality		Mean Quality	
		Total	Attempted	Closed	Closed of attempted %	Identity %	QV	Identity %	QV
CHM13	Canu	341	309	287	92.88	99.22	21.07	98.93	19.7
	Flye	341	227	202	88.98	97.54	16.09	97.51	16.03
	Shasta	341	94	92	97.87	99.47	22.74	99.37	21.99
	Wtdbg2	341	70	62	88.57	99.36	21.96	99.28	21.43
HG00733	Canu	179	137	124	90.51	98.73	18.95	98.43	18.05
	Flye	179	98	80	81.63	98.09	17.18	97.76	16.49
	Shasta	179	42	40	95.23	98.76	19.08	98.13	17.30
	Wtdbg2	179	52	46	88.46	98.70	18.87	98.02	17.04

Table D.7: BAC analysis intersection of attempted BACs by all four assemblers, 65 on CHM13 and 27 on HG00733. *Closed* refers to the number of BACs for which 99.5% of their length aligns to a single locus. *Attempted* refers to the number of BACs which have an alignment for > 55Kb of sequence with > 90% identity to only one contig (BACs which have such alignments to multiple contigs are excluded). Identity metrics are for *closed* BACs.

Sample	Assembler Polisher	BAC counts				Median Quality		Mean Quality	
		Total	Attempted	Closed	Closed of attempted %	Identity %	QV	Identity %	QV
CHM13	Canu	65	65	64	98.50	99.29	21.53	99.21	21.01
	Flye	65	65	65	100.00	97.57	16.16	97.61	16.22
	Shasta	65	65	65	100.00	99.50	23.03	99.41	22.33
	Wtdbg2	65	65	59	90.80	99.39	22.17	99.29	21.49
HG00733	Canu	27	27	26	96.30	98.66	18.76	98.54	18.37
	Flye	27	27	27	100.00	98.07	17.14	98.08	17.16
	Shasta	27	27	27	100.00	98.80	19.23	98.30	17.71
	Wtdbg2	27	27	26	96.30	98.75	19.01	98.53	18.32

Table D.8: Base-level accuracies on four different assemblers for three samples. Analysis is performed with whole-genome truth sequences.

Sample	Assembler	Percentage Errors			
		Balanced	Identity	Deletion	Insertion
HG002 Guppy 2.3.5	Shasta	0.975%	0.061%	0.849%	0.065%
	Wtdbg2	1.181%	0.080%	1.073%	0.029%
	Canu	1.400%	0.065%	1.316%	0.020%
	Flye	1.636%	0.068%	0.450%	1.118%
HG00733 Guppy 2.3.5	Shasta	1.062%	0.083%	0.887%	0.093%
	Wtdbg2	1.217%	0.108%	1.059%	0.051%
	Canu	1.328%	0.074%	1.224%	0.031%
	Flye	1.854%	0.089%	0.445%	1.320%
CHM13 Guppy 2.3.1	Shasta	0.540%	0.039%	0.430%	0.072%
	Wtdbg2	0.689%	0.068%	0.583%	0.038%
	Canu	0.705%	0.038%	0.643%	0.024%
	Flye	2.213%	0.051%	0.448%	1.715%

Table D.9: Base-level accuracies on four different assemblers for three samples in the regions of intersection of the assemblies. Analysis is performed only on regions where all assemblers have an assembled sequence.

Sample	Assembler	Percentage Errors			
		Balanced	Identity	Deletion	Insertion
HG002 Guppy 2.3.5	Shasta	0.943%	0.056%	0.823%	0.064%
	Wtdbg2	1.145%	0.077%	1.041%	0.028%
	Canu	1.319%	0.050%	1.253%	0.016%
	Flye	1.554%	0.063%	0.432%	1.059%
HG00733 Guppy 2.3.5	Shasta	1.021%	0.064%	0.875%	0.083%
	Wtdbg2	1.162%	0.088%	1.034%	0.041%
	Canu	1.307%	0.065%	1.213%	0.030%
	Flye	1.847%	0.068%	0.431%	1.348%
CHM13 Guppy 2.3.1	Shasta	0.513%	0.016%	0.406%	0.048%
	Wtdbg2	0.660%	0.054%	0.575%	0.030%
	Canu	0.692%	0.027%	0.645%	0.021%
	Flye	2.198%	0.036%	0.460%	1.702%

Table D.10: Runtime and cost of three assembly workflows on Amazon Web Services (AWS) platform.

Method	Sample	Minutes	Threads Used	Peak Memory	AWS Instance Type	AWS Instance Cost
WTDBG2	HG00733	2971	63	365	r5a.16xlarge	$3.62
	GM24385	1752	63	293	r5a.16xlarge	$3.62
	CHM13	1655	63	312	r5a.16xlarge	$3.62
WTDBG2 (wtpoa-cns)	HG00733	248	31	12	r5a.16xlarge	$3.62
	GM24385	274	24	12	r5a.16xlarge	$3.62
	CHM13	257	31	12	r5a.16xlarge	$3.62
Flye	HG00733	3421	123	1013	x1.32xlarge	$13.34
	GM24385	3749	64	727	x1.16xlarge	$6.67
	CHM13	4084	126	911	x1.32xlarge	$13.34
Shasta	HG00733	298	128	966	x1.32xlarge	$13.34
	HG01109	355	128	-	x1.32xlarge	$13.34
	HG01243	296	128	-	x1.32xlarge	$13.34
	HG02055	309	128	-	x1.32xlarge	$13.34
	HG02080	276	128	-	x1.32xlarge	$13.34
	HG02723	373	128	-	x1.32xlarge	$13.34
	HG03098	238	128	-	x1.32xlarge	$13.34
	HG03492	200	128	-	x1.32xlarge	$13.34
	GM24385	240	128	692	x1.32xlarge	$13.34
	GM24149	427	128	-	x1.32xlarge	$13.34
	GM24143	451	128	-	x1.32xlarge	$13.34
	CHM13	317	128	-	x1.32xlarge	$13.34

Table D.11: Runtime breakdown for each step of the Shasta assembler.

Sample	Input	MinHash	Alignments	Marker graph creation	Transitive reduction	Assemble	Output	Other	Total
HG00733	30	9	93	73	17	15	2	55	298
HG01109	29	10	136	89	16	17	2	53	355
HG01243	23	7	104	73	16	15	2	51	296
HG02055	25	9	113	73	15	15	2	53	309
HG02080	22	7	95	67	15	14	2	49	276
HG02723	29	9	146	89	19	16	2	59	373
HG03098	23	8	73	53	14	14	2	47	238
HG03492	19	7	57	44	11	14	2	40	200
GM24385	20	7	92	49	12	13	2	41	240
GM24149	34	11	149	124	21	18	2	64	427
GM24143	35	11	168	120	24	18	2	69	451
CHM13	21	6	173	67	12	13	2	46	345
Average	26	8	117	77	16	15	2	52	317
Percent of total	8%	3%	37%	24%	5%	5%	1%	17%	100%

Table D.12: Structural variants extracted from HG002 assembly graph compared to GIAB SV set in high-confidence regions.

Metric	HG002					
	TP	FP	FN	Precision	Recall	F_1
Total	2961	1580	1202	0.6521	0.7117	0.6806
Inserts	2152	1203	810	0.6414	0.7117	0.7289
Deletes	809	377	392	0.6821	0.6681	0.6750

Contiguously assembling MHC haplotypes

Table D.13: CHM13 MHC unpolished Shasta assembly as compared to the nearest matching haplotype in hg38 (GL000251.2)

Assembler	Best Contig	Disagreements	Largest Aligned	Mismatch Rate	Indel Rate
Shasta	62	6	2,788,362	0.00296	0.00399
Canu	tig00589784	5	2,792,139	0.00331	0.00607
Flye	contig_115	6	2,787,570	0.00543	0.01106
wtdbg2	ctg25	32	1,819,753	0.00553	0.00576

Table D.14: QUAST results for the HG00733 trio-binned maternal reads, using all four assemblers.

Metric	HG00733-Mother			
	Shasta	Wtdbg2	Flye (initial)	Canu
# contigs	1,934	4,028	1,634	877
Total length	2,754,225,214	2,690,619,717	2,791,893,188	2,829,920,708
N50	9,071,623	14,125,235	25,658,831	19,451,828
NG50	7,702,138	10,217,387	23,775,989	16,507,795
# disagreements	705	3,661	6,082	2,161
Genome fraction (%)	90.824	87.373	92.121	92.298
Duplication ratio	0.993	0.996	0.982	0.999
# mismatches per 100 kbp	194.15	287.89	549.61	232.72
# indels per 100 kbp	576.55	859.83	1585.30	724.67
Total aligned length	2,748,135,723	2,650,821,801	2,751,532,754	2,798,797,021
NA50	7,805,090	7,615,651	15,615,208	11,947,316
NGA50	6,339,949	5,584,544	12,833,996	10,085,023

Table D.15: HG00733 Maternal trio binned MHC unpolished Shasta assembly as compared to the nearest matching haplotype in hg38 (GL000255.1)

Assembler	Best Contig	Disagreements	Largest Aligned	Mismatch Rate	Indel Rate
Shasta	226	0	4,289,729	0.00206	0.00538
Canu	tig00002130	0	4,289,729	0.00182	0.00676
Flye	contig_295	0	4,289,729	0.00579	0.01759
wtdbg2	ctg36	23	1,418,939	0.00592	0.00905

Deep neural network based polishing achieves QV30 long-read only polishing accuracy

Table D.16: Base-level accuracies comparing Racon & Medaka and MarginPolish & HELEN pipelines on Shasta assemblies for three samples. Analysis is performed with whole-genome truth sequences.

Sample	Polisher		Percentage Errors			
	Method	Model	Balanced	Identity	Deletion	Insertion
HG002 Guppy 2.3.5	Shasta	Unpolished	0.975%	0.061%	0.849%	0.065%
	Racon	4x	0.665%	0.054%	0.579%	0.032%
	Medaka	r941_flip235	0.393%	0.051%	0.303%	0.039%
	MarginPolish	guppy_ff235	0.372%	0.043%	0.248%	0.081%
	HELEN	rl941_flip235	0.279%	0.038%	0.171%	0.070%
HG00733 Guppy 2.3.5	Shasta	Unpolished	1.062%	0.083%	0.887%	0.093%
	Racon	4x	0.715%	0.080%	0.570%	0.066%
	Medaka	r941_flip235	0.455%	0.075%	0.311%	0.069%
	MarginPolish	guppy_ff235	0.460%	0.063%	0.278%	0.118%
	HELEN	rl941_flip235	0.388%	0.066%	0.202%	0.120%
CHM13 Guppy 2.3.1	Shasta	Unpolished	0.540%	0.039%	0.430%	0.072%
	Racon	4x	0.367%	0.037%	0.199%	0.131%
	Medaka	r941_flip213	0.329%	0.033%	0.037%	0.259%
	MarginPolish	guppy_ff233	0.281%	0.027%	0.071%	0.184%
	HELEN	rl941_flip233	0.206%	0.027%	0.062%	0.117%

Table D.17: QUAST results for the Shasta assemblies for all samples, post polishing with MarginPolish-HELEN.

Sample	# contigs	Total length	N50	NG50	# mis-assemblies	Genome fraction (%)	# mismatches per 100 kbp	# indels per 100 kbp	Total aligned length	NA50	NGA50
GM24143	2,042	2,802,437,249	23,531,777	19,936,924	970	95.025	128.63	142.77	2,794,379,803	16,323,510	13,840,294
GM24149	2,368	2,816,566,939	20,798,256	17,752,973	990	95.416	130.54	134.60	2,806,847,428	13,174,778	12,128,076
GM24385	1,685	2,819,474,365	23,520,830	20,346,145	960	95.609	127.44	152.17	2,810,951,083	16,200,287	14,315,298
HG00733	1,962	2,800,357,697	24,600,414	21,701,762	877	94.976	126.23	137.92	2,792,792,711	16,156,822	12,971,070
HG01109	2,111	2,820,988,852	21,532,001	18,279,481	1,033	95.564	136.51	140.59	2,811,696,923	13,162,850	12,012,786
HG01243	1,936	2,819,065,027	22,753,128	20,884,160	920	95.521	137.50	143.02	2,810,262,570	16,040,951	14,115,348
HG02055	1,903	2,819,836,390	17,485,643	16,302,857	971	95.592	142.23	162.43	2,810,300,557	13,840,319	12,123,357
HG02080	1,814	2,803,471,776	18,701,305	15,584,440	920	95.045	128.16	134.35	2,794,749,368	12,401,739	11,561,569
HG02723	1,813	2,805,268,038	25,163,327	20,265,678	1,110	95.062	143.30	147.09	2,796,332,696	15,390,923	13,175,818
HG03098	1,790	2,811,295,217	22,571,315	19,620,076	986	95.395	144.36	170.40	2,802,844,336	14,045,283	12,089,849
HG03492	1,811	2,811,690,127	24,629,163	22,891,947	854	95.364	126.61	147.22	2,804,103,412	16,317,390	12,930,516
CHM13	1,186	2,819,245,173	46,206,794	41,255,275	1,107	95.281	136.58	140.38	2,808,536,514	23,540,225	19,532,176

Table D.18: Base-level accuracies comparing Racon & Medaka and MarginPolish & HELEN pipelines against CHM13 Chromosome-X. The truth Chromosome-X sequence used reflects the most accurate haploid truth sequence available.

Sample	Polisher		Percentage Errors			
	Method	Model	Balanced	Identity	Deletion	Insertion
CHM-13 Chromosome-X	Shasta	Unpolished	0.469%	0.014%	0.404%	0.051%
	Racon	4x	0.313%	0.017%	0.192%	0.104%
	Medaka	r941_flip213	0.110%	0.012%	0.035%	0.063%
	MarginPolish	guppy_ff233	0.215%	0.008%	0.055%	0.153%
	HELEN	rl941_flip233	0.143%	0.007%	0.041%	0.095%
		rl941_flip231	0.064%	0.006%	0.036%	0.022%

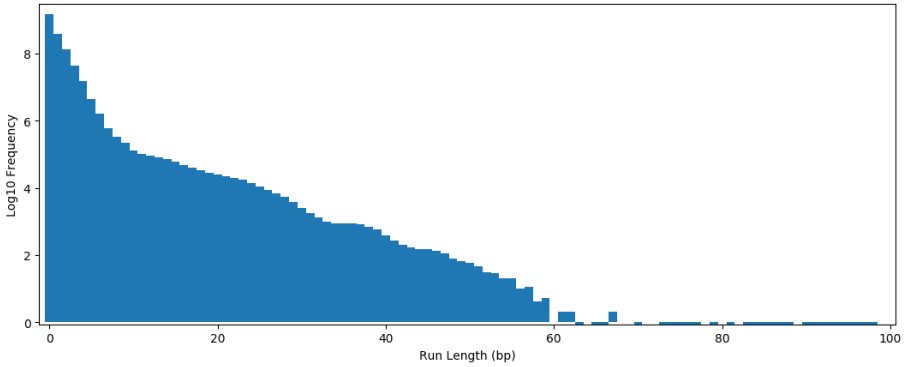

Figure D.2: Log frequency of each run length as found in the GRCh38 reference for all bases A,C,G,T up to 100bp. Run lengths greater than 15 account for approximately 0.012% of all homopolymer runs in GRCh38.

Table D.19: Base-level accuracies improvements with MarginPolish and HELEN pipeline on four different assemblers for two samples. Analysis is performed with whole-genome truth sequences.

Sample	Polisher		Percentage Errors			
	Method	Model	Balanced	Identity	Deletion	Insertion
HG00733 Guppy 2.3.5	Shasta	Unpolished	1.062%	0.083%	0.887%	0.093%
	MarginPolish	guppy_ff235	0.460%	0.063%	0.278%	0.118%
	HELEN	rl941_flip235	0.388%	0.066%	0.202%	0.120%
	Wtdbg2	Unpolished	1.217%	0.108%	1.059%	0.051%
	MarginPolish	guppy_ff235	0.538%	0.083%	0.333%	0.122%
	HELEN	rl941_flip235	0.473%	0.089%	0.257%	0.127%
	Canu	Unpolished	1.328%	0.074%	1.224%	0.031%
	MarginPolish	guppy_ff235	0.438%	0.050%	0.290%	0.098%
	HELEN	rl941_flip235	0.355%	0.050%	0.206%	0.099%
	Flye	Unpolished	1.854%	0.089%	0.445%	1.320%
	MarginPolish	guppy_ff235	0.425%	0.062%	0.257%	0.106%
	HELEN	rl941_flip235	0.356%	0.064%	0.183%	0.109%
CHM13 Guppy 2.3.1	Shasta	Unpolished	0.540%	0.039%	0.430%	0.072%
	MarginPolish	guppy_ff233	0.281%	0.027%	0.071%	0.184%
	HELEN	rl941_flip233	0.206%	0.027%	0.062%	0.117%
	Wtdbg2	Unpolished	0.689%	0.068%	0.583%	0.038%
	MarginPolish	guppy_ff233	0.361%	0.049%	0.112%	0.201%
	HELEN	rl941_flip233	0.296%	0.053%	0.115%	0.129%
	Canu	Unpolished	0.705%	0.038%	0.643%	0.024%
	MarginPolish	guppy_ff233	0.255%	0.013%	0.075%	0.168%
	HELEN	rl941_flip233	0.173%	0.012%	0.058%	0.103%
	Flye	Unpolished	2.213%	0.051%	0.448%	1.715%
	MarginPolish	guppy_ff233	0.256%	0.022%	0.058%	0.176%
	HELEN	rl941_flip233	0.185%	0.024%	0.052%	0.109%

Table D.20: Single-chromosome error rates after polishing with short reads. 10X Chromium reads for sample CHM13 were used to polish via Pilon polishing software. The top half of the table shows the results of three rounds of Pilon, starting from the CHM13 Shasta chrX assembly that had been polished with MarginPolish and HELEN. The bottom half shows the results of three rounds of Pilon, starting from the raw Shasta assembly.

Sample	Assembly	Percentage Errors				Q Scores			
		Balanced	Identity	Deletion	Insertion	Balanced	Identity	Deletion	Insertion
CHM13 ChrX	Shasta (polished)	0.064%	0.006%	0.036%	0.022%	31.92	42.40	34.42	36.51
	Pilon 1x	0.025%	0.004%	0.012%	0.008%	36.06	43.75	39.16	40.75
	Pilon 2x	0.023%	0.004%	0.012%	0.007%	36.29	43.51	39.32	41.34
CHM13 ChrX	Shasta (raw)	0.468%	0.014%	0.404%	0.051%	23.29	38.57	23.94	32.95
	Pilon 1x	0.449%	0.011%	0.395%	0.043%	23.48	39.78	24.03	33.68
	Pilon 2x	0.425%	0.011%	0.373%	0.041%	23.71	39.49	24.29	33.84

Table D.21: Runtime and cost of two polishing workflows on Amazon Web Services (AWS) platform.

Method	Sample	Minutes	Threads Used	Peak Memory	Instance Type	Instance Cost
Racon (4x)	HG00733	3099	62	574	r5a.24xlarge	$5.42
	GM24385	2342	62	501	r5a.24xlarge	$5.42
	CHM13	3700	62	281	r5a.24xlarge	$5.42
Medaka mini_align	HG00733	611	62	101	c5.18xlarge	$3.06
	GM24385	489	62	115	c5.18xlarge	$3.06
	CHM13	810	60	143	c5.18xlarge	$3.06
Medaka call_consensus	HG00733	8611	62	164	c5n.18xlarge	$3.89
	GM24385	3355	62	150	c5n.18xlarge	$3.89
	CHM13	2532	62	149	c5n.18xlarge	$3.89
MarginPolish	HG00733	680	90	66	m5.metal	$4.61
	HG01109	912	70	57	c5.18xlarge	$3.06
	HG01243	835	70	65	c5.18xlarge	$3.06
	HG02055	733	70	77	c5.18xlarge	$3.06
	HG02080	793	70	64	c5.18xlarge	$3.06
	HG02723	1000	64	60	c5.18xlarge	$3.06
	HG03098	852	70	78	c5.18xlarge	$3.06
	HG03492	777	70	80	c5.18xlarge	$3.06
	GM24385	842	70	66	c5.18xlarge	$3.06
	GM24149	1037	64	103	c5.18xlarge	$3.06
	GM24143	1051	64	84	c5.18xlarge	$3.06
	CHM13	739	70	65	c5.18xlarge	$3.06
HELEN consensus	HG00733	216	8 GPUs	-	p2.8xlarge	$7.20
	HG01109	204	8 GPUs	-	p2.8xlarge	$7.20
	HG01243	233	8 GPUs	-	p2.8xlarge	$7.20
	HG02080	212	8 GPUs	-	p2.8xlarge	$7.20
	HG03098	216	8 GPUs	-	p2.8xlarge	$7.20
	GM24385	208	8 GPUs	-	p2.8xlarge	$7.20
	GM24143	226	8 GPUs	-	p2.8xlarge	$7.20
HELEN stitch	HG00733	59	32	-	p2.8xlarge	$7.20
	HG01109	50	32	-	p2.8xlarge	$7.20
	HG01243	49	32	-	p2.8xlarge	$7.20
	HG02080	54	32	-	p2.8xlarge	$7.20
	HG03098	65	32	-	p2.8xlarge	$7.20
	GM24385	68	32	-	p2.8xlarge	$7.20
	GM24143	62	32	-	p2.8xlarge	$7.20

Table D.22: Runtime and cost of two polishing workflows run on a 29 Mb contig from the HG00733 Shasta assembly. MarginPolish uses an improved stitch method not used in original runs and Racon was run once instead of four times as was done in the full runs. All runs were configured to use 32 CPUs, except for the GPU runs which were performed with 16 CPUs and 1 GPU (Tesla P100).

Application	**Runtimes**	**Avg Runtime**
MarginPolish	16.6	16.46
	16.47	
	16.31	
HELEN consensus (CPU)	97.46	95.86
	95.55	
	94.56	
HELEN consensus (GPU)	1.63	1.67
	1.72	
	1.65	
HELEN stitch	0.76	0.78
	0.78	
	0.80	
Racon 1x	52.00	52.04
	52.15	
	51.98	
mini_align	3.01	3.00
	3.00	
	2.98	
Medaka (CPU)	17.26	17.01
	16.78	
	16.98	
Medaka consensus (GPU)	10.55	10.62
	10.73	
	10.57	
Medaka stitch (GPU)	0.68	0.68
	0.68	
	0.68	

Table D.23: Transcript-level analysis with Comparative Annotation Toolkit (CAT) of MarginPolish & HELEN and Racon & Medaka on three samples from Shasta assemblies.

Metric		HG002		HG00733		CHM13	
		HELEN	MEDAKA	HELEN	MEDAKA	HELEN	MEDAKA
Transcripts Found	Total	83093	83105	83002	82928	82833	82807
	Percent	99.536	99.551	99.427	99.339	99.225	99.194
Full mRNA Coverage	Total	25721	20367	28612	26573	40132	38081
	Percent	30.811	24.397	34.274	31.832	48.074	45.617
Full CDS Coverage	Total	41396	36248	45104	43956	53089	52297
	Percent	49.588	43.421	54.030	52.655	63.595	62.646
Transcripts With Frameshift	Total	35339	40783	31333	32647	23261	24441
	Percent	42.332	48.854	37.534	39.108	27.864	29.278
Transcripts With Original Introns	Total	76880	76883	76618	76463	76807	76803
	Percent	92.094	92.098	91.780	91.594	92.006	92.002
Transcripts With Full CDS Coverage	Total	41396	36248	45104	43956	53089	52297
	Percent	49.588	43.421	54.030	52.655	63.595	62.646
Transcripts With Full CDS Coverage And No Frameshifts	Total	41245	36158	44982	43860	52966	52160
	Percent	49.407	43.313	53.884	52.540	63.448	62.482
Transcripts With Full CDS Coverage And No Frameshifts And Original Introns	Total	41021	35952	44692	43546	52616	51807
	Percent	49.139	43.067	53.536	52.163	63.028	62.059

Table D.24: Gene-level analysis with Comparative Annotation Toolkit (CAT) of MarginPolish & HELEN and Racon & Medaka on three samples from Shasta assemblies.

Metric		HG002		HG00733		CHM13	
		HELEN	MEDAKA	HELEN	MEDAKA	HELEN	MEDAKA
Genes Found	Total	19536	19531	19537	19511	19505	19490
	Percent	99.268	99.243	99.273	99.141	99.111	99.035
Genes With Frameshift	Total	10933	12165	9941	10081	7300	7564
	Percent	55.554	61.814	50.513	51.225	37.093	38.435
Genes With Original Introns	Total	18212	18198	18151	18113	18217	18202
	Percent	92.541	92.47	92.231	92.038	92.566	92.49
Genes With Full CDS Coverage	Total	11070	10066	11812	11756	13648	13534
	Percent	56.25	51.148	60.02	59.736	69.35	68.77
Genes With Full CDS Coverage And No Frameshifts	Total	12454	11570	13127	13081	14625	14562
	Percent	63.283	58.791	66.702	66.468	74.314	73.994
Genes With Full CDS Coverage And No Frameshifts And Original Introns	Total	12422	11539	13098	13042	14603	14531
	Percent	63.12	58.633	66.555	66.27	74.202	73.836
Missing Genes	Total	144	149	143	169	175	190
	Percent	0.732	0.757	0.727	0.859	0.889	0.965

Table D.25: Transcript-level analysis with Comparative Annotation Toolkit (CAT) of four HG00733 assemblies polished with MarginPolish and HELEN.

Metric		HG00733			
		Flye HELEN	Canu HELEN	Wtdbg2 HELEN	Shasta HELEN
Transcripts Found	Total	83267	83334	81484	82974
	Percent	99.745	99.825	97.609	99.394
Full mRNA Coverage	Total	33078	28488	28889	30378
	Percent	39.624	34.126	34.606	36.390
Full CDS Coverage	Total	41396	44877	45321	46965
	Percent	59.754	53.758	54.290	56.259
Transcripts With Frameshift	Total	27293	32230	29525	29657
	Percent	32.694	38.608	35.368	35.526
Transcripts With Original Introns	Total	77412	77583	74683	76613
	Percent	92.731	92.936	89.462	91.774
Transcripts with Full CDS Coverage	Total	49883	44877	45321	46965
	Percent	59.754	53.758	54.290	56.259
Transcripts with Full CDS Coverage And No Frameshifts	Total	49766	44737	45217	46802
	Percent	59.614	53.590	54.165	56.064
Transcripts with Full CDS Coverage And No Frameshifts And Original Introns	Total	49459	44412	44924	46505
	Percent	59.247	53.201	53.814	55.708

Table D.26: Gene-level analysis with Comparative Annotation Toolkit (CAT) of four HG00733 assemblies polished with MarginPolish and HELEN

Metric		HG00733			
		Flye HELEN	Canu HELEN	Wtdbg2 HELEN	Shasta HELEN
Genes Found	Total	19563	19629	19174	19528
	Percent	99.405	99.741	97.429	99.228
Genes With Frameshift	Total	8698	10160	9323	9464
	Percent	44.197	51.626	47.373	48.089
Genes With Original Introns	Total	18345	18460	17709	18154
	Percent	93.216	93.801	89.985	92.246
Genes With Full CDS Coverage	Total	12966	11889	11817	12207
	Percent	65.884	60.412	60.046	62.027
Genes With Full CDS Coverage And No Frameshifts	Total	14145	13221	13047	13419
	Percent	71.875	67.18	66.296	68.186
Genes With Full CDS Coverage And No Frameshifts And Original Introns	Total	14124	13193	13017	13396
	Percent	71.768	67.038	66.143	68.069
Missing Genes	Total	117	51	506	152
	Percent	0.595	0.259	2.571	0.772

Table D.27: BUSCO results of three samples using two polishing workflows on Shasta assemblies.

Sample	Metric	Shasta MarginPolish HELEN	Shasta Racon (4x) Medaka
HG00733	Complete BUSCOs (C)	87.20%	87.10%
	Complete and single-copy BUSCOs (S)	84.20%	83.80%
	Complete and duplicated BUSCOs (D)	3.00%	3.30%
	Fragmented BUSCOs (F)	4.60%	5.30%
	Missing BUSCOs (M)	8.20%	7.60%
HG002	Complete BUSCOs (C)	89.40%	88.80%
	Complete and single-copy BUSCOs (S)	84.80%	85.80%
	Complete and duplicated BUSCOs (D)	4.60%	3.00%
	Fragmented BUSCOs (F)	3.60%	4.30%
	Missing BUSCOs (M)	7.00%	6.90%
CHM13	Complete BUSCOs (C)	86.50%	86.80%
	Complete and single-copy BUSCOs (S)	82.50%	82.80%
	Complete and duplicated BUSCOs (D)	4.00%	4.00%
	Fragmented BUSCOs (F)	5.90%	5.30%
	Missing BUSCOs (M)	7.60%	7.90%

Table D.28: BUSCO results for four assemblers on HG00733, post polishing with MarginPolish and HELEN.

Metric	HG00733			
	Flye	Canu	Wtdbg2	Shasta
Complete BUSCOs (C)	87.50%	89.80%	85.80%	87.20%
Complete and single-copy BUSCOs (S)	84.50%	86.80%	82.20%	84.20%
Complete and duplicated BUSCOs (D)	3.00%	3.00%	3.60%	3.00%
Fragmented BUSCOs (F)	5.30%	3.00%	6.30%	4.60%
Missing BUSCOs (M)	7.20%	7.20%	7.90%	8.20%

Comparing to a PacBio HiFi Assembly

Table D.29: CHM13 QUAST results for Shasta, MarginPolish, HELEN and PacBio HiFi assembly. Stratified disagreement counts were added after manual determination.

Metric	CHM13	
	Nanopore Shasta MarginPolish, HELEN	PacBio-HiFi Canu Racon
# contigs	1622	5206
Total length	2819245173	3031026325
N50	46206794	29522819
NG50	41255275	29092230
# disagreements	1107	8666
# disagreements outside Centromeres	801	2999
# disagreements outside centromeres and Seg Dups	314	893
Genome fraction (%)	95.281	97.030
# mismatches per 100 kbp	136.58	274.84
# indels per 100 kbp	140.38	32.99
Total aligned length	2808536514	2954558720
NA50	23540225	20440378
NGA50	19532176	20029136

Table D.30: Disagreement count in the intersection of the assemblies between the PacBio-HiFi and the Shasta assembly of CHM13. Total Disagreements is all disagreements found in 100bp before windows before taking the intersection, note it is very close to that reported by QUAST. Consensus disagreements: Disagreements in the intersection of the four assemblies.

Sample	Assembler	Total disagreements	Consensus disagreements
CHM13	PacBio-HiFi	8469	594
	Shasta	1073	380

Table D.31: CHM13 Chromosome-X error rate analysis with Pomoxis for Shasta, MarginPolish, HELEN, and PacBio HiFi assembly.

Sample	Sequencing Platform	Method		Percentage errors			
		Assembler	Polisher	Balanced	Identity	Deletion	Insertion
CHM13 Chr-X	PacBio HiFi	Canu	Racon	0.008%	0.001%	0.004%	0.003%
	Nanopore	Shasta	MarginPolish & HELEN	0.064%	0.006%	0.036%	0.022%

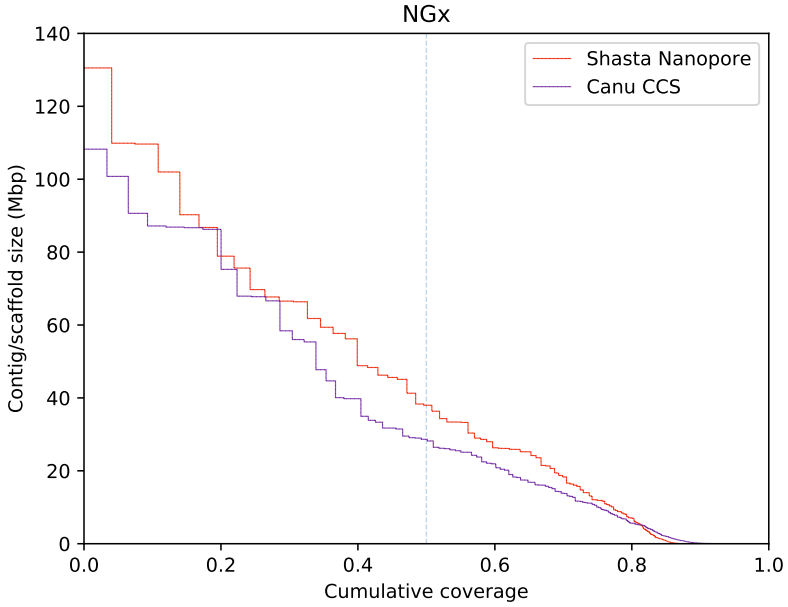

Figure D.3: Contig NGx for CHM13 Shasta-HELEN nanopore assembly vs Canu CCS (HiFi) assembly

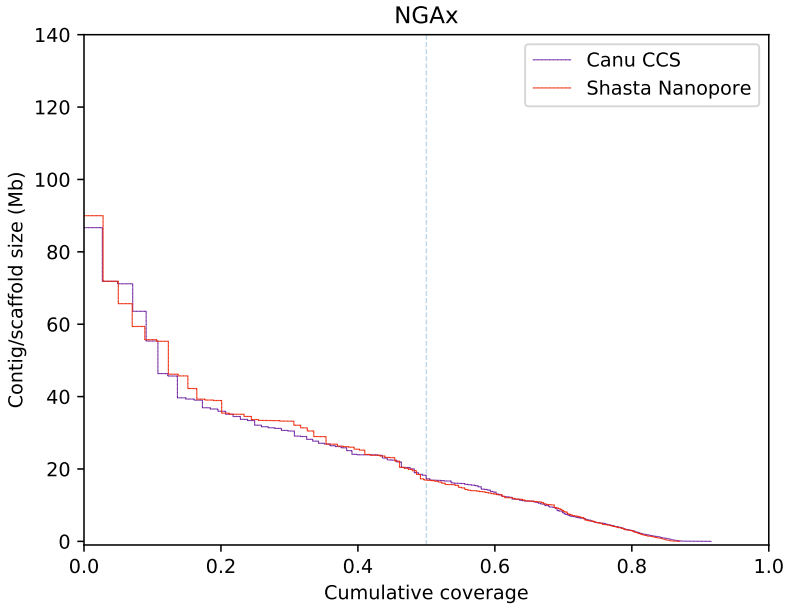

Figure D.4: Contig NGAx for CHM13 Shasta-HELEN nanopore assembly vs Canu CCS (HiFi) assembly

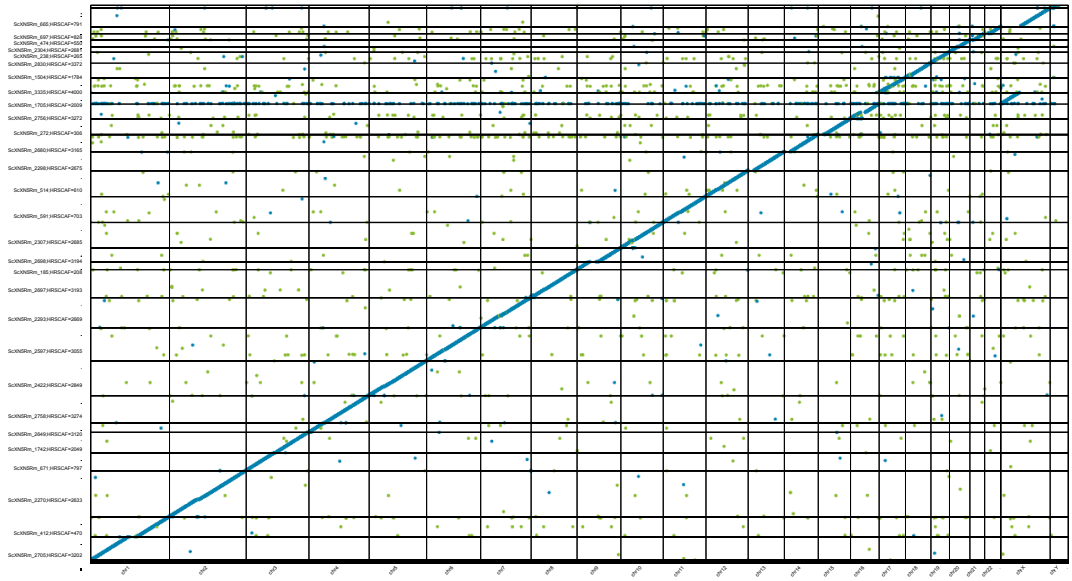

Figure D.5: Dotplot for the scaffolded HG002 assembly, aligned with GRCh38. Blue dots represent unique alignments and orange dots represent repetitive alignments.

Assembling, polishing and scaffolding 11 human genomes at near chromosome scale

Table D.32: QUAST results for all 11 Shasta assemblies scaffolded with HiRise, post polishing with MarginPolish-HELEN

Sample	# contigs	Total length	N50	NG50	# mis-assemblies	# scaffold gap extensive mis-assembies	Genome fraction (%)	# mismatches per 100 kbp	# indels per 100 kbp	Total aligned length	NA50	NGA50
GM24143	1,184	2,802,523,049	129,960,437	128,216,303	1,466	4	95.027	128.28	142.79	2,792,775,664	20,657,530	16,966,477
GM24149	1,323	2,816,683,224	129,643,816	128,275,807	1,530	11	95.417	130.24	134.58	2,804,735,382	18,446,390	15,435,923
GM24385	1,019	2,819,527,260	118,169,209	102,591,941	1,335	6	95.606	127.19	152.25	2,809,570,528	22,369,161	16,601,924
HG00733	1,056	2,800,455,909	129,857,865	118,785,172	1,337	8	94.974	126.16	138.09	2,791,610,554	22,141,375	17,570,210
HG01109	1,156	2,821,098,626	130,282,751	130,166,418	1,529	5	95.559	136.73	140.63	2,809,413,640	19,932,703	17,228,023
HG01243	1,006	2,819,162,443	128,571,344	118,762,399	1,381	7	95.517	137.47	143.03	2,808,041,766	22,146,722	17,559,055
HG02055	977	2,819,933,140	130,184,428	128,180,737	1,387	8	95.587	141.91	162.46	2,809,195,864	21,057,279	18,446,049
HG02080	934	2,803,570,658	129,931,575	128,451,196	1,470	9	95.041	127.98	134.36	2,793,854,132	20,418,609	16,379,851
HG02723	982	2,805,356,030	130,365,062	128,975,828	1,499	9	95.06	143.45	147.13	2,794,747,200	20,232,566	17,865,825
HG03098	926	2,811,385,538	130,040,472	128,535,908	1,439	4	95.391	144.36	170.40	2,801,774,564	22,165,948	17,439,948
HG03492	901	2,811,782,250	130,277,907	100,251,163	1,381	7	95.362	126.54	147.23	2,803,106,787	20,001,587	16,836,756