UNIVERSITY OF CALIFORNIA
SANTA CRUZ

METHODS FOR NANOPORE GENOME ASSEMBLY AND PHASING

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

DOCTOR OF PHILOSOPHY

in

BIOMOLECULAR ENGINEERING AND BIOINFORMATICS

by

Ryan M. Lorig-Roach

March 2023

The Dissertation of Ryan M. Lorig-Roach
is approved:

Professor David Haussler, Chair

Professor Benedict Paten, Co-advisor

Professor Karen Miga

Professor Beth Shapiro

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

Abstract

Methods for nanopore genome assembly and phasing

by

Ryan M. Lorig-Roach

In this work, I present methods which aim to facilitate the generation of diverse genomic assembly data, using primarily nanopore sequencing and the Shasta assembler. First, I describe methods for improving consensus quality in nanopore assemblies. Second, I present a method for transforming graph representations of assemblies. Finally, I describe a graph based phasing method which enables accurate, de novo, chromosome-scale phasing, with a total of 2 flow cells of nanopore sequence.

Acknowledgments

I thank my committee and colleagues, for their trust and patience while I developed as a computational scientist. Many thanks to my friends and family who convinced me it was attainable and supported me throughout. Finally, I have a sincere gratitude for David Deamer's willingness to mentor me at an early stage in my career, and whose infectious passion for science and discovery inspired me and encouraged me to pursue research.

Part I

Introduction

Introduction

Early biochemical experiments used a combination of extracts from infectious cultures[1] to demonstrate that DNA conferred heritability of physical traits. Soon after, it was learned that variation in a single protein complex is directly responsible for sickle cell anemia[2], which was the first time that protein structure was directly linked to a medical condition. The full implication of DNA and its link to phenotype was later established with the discovery of mRNA[3]. As an indication of its significance to our understanding of life, the flow of information between DNA, RNA, and protein was dubbed the "central dogma" of biology[4]. Future research then began to map observable traits directly to DNA, and the first medically relevant DNA mutation was mapped in 1983 using genealogy and cloning[5]. This process of mapping traits to genetic loci inevitably motivated more ambitious projects which aimed to learn the complete sequence of the human genome, as it would be a necessary first step to fully understanding life, and would greatly accelerate our ability to diagnose and cure heritable diseases.

Sequencing

Having only 4 constituent components, known as nucleotides or bases, DNA can feasibly be "sequenced" by iteratively processing it in a linear fashion, to learn its sequential ordering of Adenine, Guanine, Cytosine, and Thymine. Early methods for accomplishing this were based on the principle of templated polymerization of the double stranded molecule, and have diversified into a wide variety of implementations, generally improving over the years in terms of cost, accuracy, and length of sequences.

Sanger sequencing was the first accurate and robust protocol for polymerization-based sequencing. It relies on the incorporation of a small fraction of modified A,C,G, or T base, which would randomly terminate a growing strand of DNA, producing a mixture of molecules, each with a length corresponding to the relative locus of that base[6]. Repeating this style of early termination for each base yields four series which, in combination with a size separation technique (gel electrophoresis) creates a complete ordering of all bases. This method was relatively efficient at the time, but has a length limit of around 1,000 bases, or as many bases as could be visually distinguished by length within a

single gel electrophoresis experiment. Compared to modern methods, there is a large resource and time cost associated with the polymer termination, running of the gel, and interpretation of results. A more efficient variation of this protocol which uses 4 fluorescent labels instead of four independent polymerization trials was eventually introduced.

To overcome the throughput limitations associated with chain terminating methods, pyrosequencing was developed, and eventually widely distributed by Roche and Illumina[7,8]. The main contribution of this method was that it eliminated the need for gel electrophoresis by converting polymerization byproducts into a readable light signal, using an enzyme. The polymerization reaction in this protocol is conducted in a stepwise manner, by washing fixed polymers with one of the 4 bases at a time, and the fluorescence is readable in real time. This method was automated and enabled scaling experiments to much larger quantities of DNA, but it still depended on synthesis, and large populations of DNA molecules. A common pitfall for methods such as these is the desynchronization of the stepwise reaction, which then leads to a mixture of signals read by the machine, and results in an upper length limit of around 150 base pairs (bp) on this type of sequencing[9]. Illumina's polymerization based sequencing is still widely used as a result of its efficiency and accuracy, but its read length has been greatly surpassed by recent methods which sequence single molecules[10,11].

PacBio Single Molecule Real Time Sequencing (SMRT) uses a fixed polymerase in a reaction chamber which localizes the fluorescent signal of 4 modified bases[12]. This system allows for the accurate sequencing of individual molecules, further improving efficiency and, perhaps more importantly, increasing the maximum readable sequence length from 150 bases to tens of thousands of bases. Initially this method yielded a sequence accuracy of only around 90%[9,13], which was later improved to 99.9% with an alternative method to convert double stranded DNA to a circular molecule that can be reread many times by the sequencer[14]. While extremely accurate, this method is limited by length and cost in comparison to synthesis-free methods.

Synthesis-free sequencing

Nanopore sequencing directly processes endogenous DNA strands to generate a signal which depends on the structure and order of the nucleotides contained in the strand. The instrument establishes a voltage across a membrane containing pores that narrowly accommodate single stranded DNA. When a strand passes through a pore, a sensor measures

the current, determined by the resistance of the nucleotides that occupy the pore. This process produces a continuous time series of measurements, which varies from around 50-150 picoamps[10]. The time series can then be translated to find the original sequence of nucleotides that were processed.

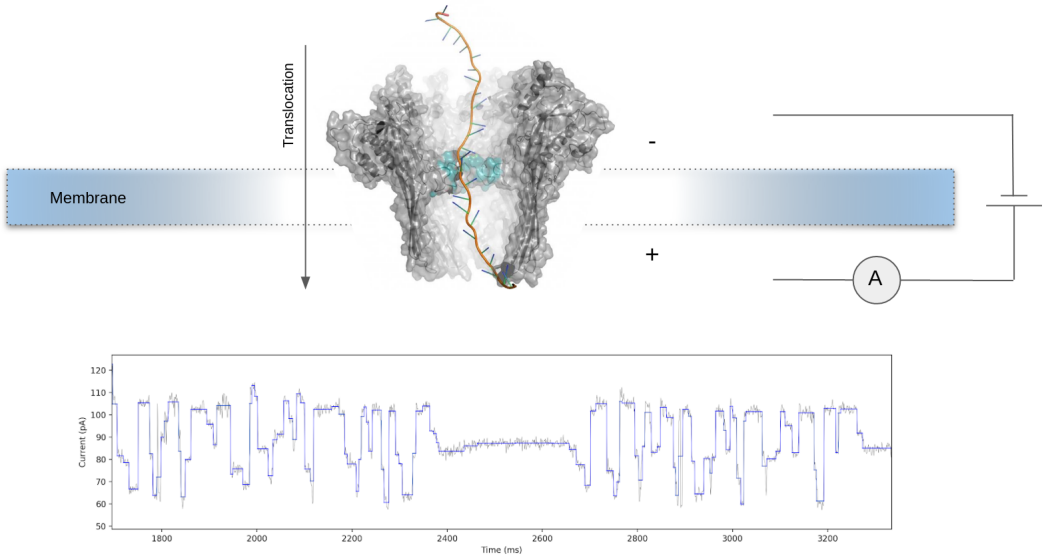

Figure 1: Schematic of nanopore translocation and an example of a real signal emitted from the device. In gray (below) is the raw signal and in blue is the output of a segmenter, which discretizes the noisy output. Note the level portion in the middle which resulted from a Thymine homopolymer.

The signal depends nonuniformly on multiple bases which occupy the pore at any given time. Therefore, the range of outputs from the system is much greater than 4 discrete values, to the extent that it is essentially a continuous distribution from 50-150pA. The system is also confounded by noise, and a stochastic rate of translocation. For these reasons, decoding the signal is not a perfect process, and the main source of error is in the insertion and deletion component of identity, rather than substitutions. The error rate of decoding or "basecalling" is on the order of 4\% for older pore designs (R9) and 1\% for new pore structures (R10).

Nanopore sequencing has unusual benefits because it directly sequences native polymers from the cell. It does not depend on artificial replication of DNA to amplify the signal. This simplifies sample preparation, and makes it possible to sequence longer reads, with lengths observed up to 2 million bases[15]. The length distribution of nanopore libraries

extends much further into the 100kb+ range than any other technology, greatly reducing complications in assembly of repetitive regions or duplicated genes.

Linked reads

Short read sequencing methods have alternative means of spanning large distances. Using adapters or barcodes during library prep, a long molecule of DNA can be sequenced by multiple short reads which are identifiable by downstream applications as having been derived from the same molecule. In Illumina sequencers this is accomplished with bridge amplification.

Bridge amplification sequences the ends of a continuous molecule of DNA, and the resulting reads are referred to as "linked" or "paired end" reads. The DNA molecule has a known length distribution, and it can be used to improve mapping of short reads by conditioning the mapping of each read by its pair and the expected gap size between them[8,16–18]. In more recent sequencing protocols, such as the 10X Genomics Chromium method, physical isolation of molecules in a reaction chamber is used to re-sequence long fragments of DNA many times over with short reads[19,20]. As with paired end methods, clustering can be used to map reads from the same molecule.

Hi-C is another approach to linked read sequencing which takes advantage of the packing of chromatin in the nucleus. It uses reversible formaldehyde crosslinking to find regions of the genome that are distant in linear sequence space, but proximal in physical space[21]. This protocol is relatively more straightforward than 10X methods for producing linked reads, because it doesn't require a specialized machine for extracting and partitioning genetic material. Instead, Hi-C uses the organization of DNA in the nucleus itself. Paired reads for this method can span tens of millions of bases, effectively providing full chromosome linkage information for genome assembly and variation detection.

Assembly

Genome assembly is the process by which chromosomal sequences are inferred from reads that are shorter than a chromosome. In general, this process relies on redundant, overlapping reads which are sampled randomly from the chromosome[22]. The initial challenge is to then find reads which share similar sequences and therefore are likely derived from a

common locus in the underlying genome. However, the process of naively comparing reads to one another would have at minimum a quadratic (½n^2) complexity as a result of the need to iterate all pairs of reads. The challenge of computing this overlap of reads is addressed primarily in two paradigms of assembly: Overlap Layout Consensus (OLC) and De Bruijn Graph assembly[23].

De Bruijn Graph assembly is an elegant solution to the overlap problem which reduces sequences into fixed length subsequences or "k-mers". In this paradigm, each read constitutes a series of k-mers which are added to a graph as nodes. Edges between k-mers are added similarly, describing the adjacency of k-mers observed in the reads. Each k-mer is only added to the graph once, and can have multiple in and out edges. The resulting graph is a succinct description of all observed sequences, and given a low enough error rate, the true genomic sequence is guaranteed to exist as a walk through this graph. However, in practice, short k-mer lengths create a challenge, because they can be shorter than duplicated regions of the genome, which creates cycles in the graph, and complicates traversal. In addition to this, error in sequencing results in spurious k-mers, which need to be filtered or ignored during traversal. The De Bruijn Graph method of assembly is well defined, and has consistently been used in a variety of assemblers [24–27].

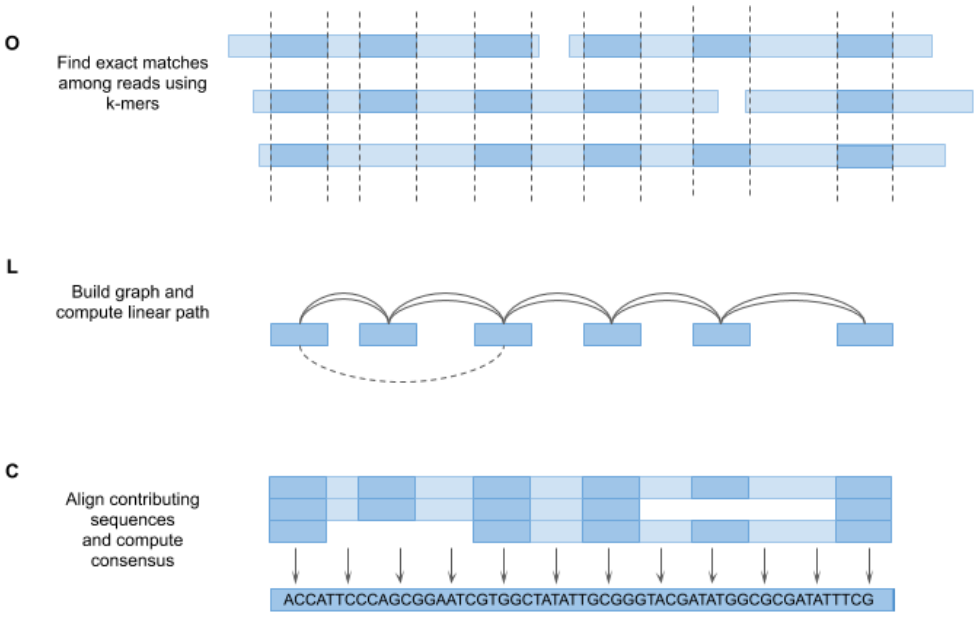

Figure 2: Each step of the Overlap Layout Consensus (OLC) method for assembly is briefly summarized. **Overlap (O):** Exact matches are found among reads that have been "clustered" together via MinHash. Alignment is performed at the level of markers, and used to build a collapsed graph representation. **Layout (L):** The graph representation is pruned and simplified. Only the minimum set of edges that spans all nodes is retained. If the graph is nonlinear it is traversed to identify linear segments. **Consensus (C):** Any edges which were not explicitly matched during overlap are aligned with multiple sequence alignment and their consensus is computed.

Overlap Layout Consensus (OLC) is an alternative approach to assembly which attempts to compute direct pairwise overlaps between reads. Generally, the challenge of quadratic overlap complexity is averted by using approximations of overlap, which again depend on fixed length subsequences[28–32]. Once an approximate overlap is computed, a more computationally expensive evaluation of overlap can be used to further refine overlap candidates, producing a graph that represents read adjacency. As with De Bruijn Graph methods, the "layout" step of OLC then attempts to traverse the overlaps in a manner that reconstructs the underlying chromosomes. Once a traversal is complete, the consensus step ensues, which uses the distribution of observed sequences at each locus to infer an consensus sequence. OLC methods are computationally more flexible and tunable than De Bruijn graph methods, but they often depend on heuristics, which can make implementations more complex.

Assembling repetitive content

The human genome is more than 50\% repetitive DNA[33,34]. Repetitive elements can vary in length from several to millions of bases[35]. In the case of NOTCH2NL, for example, hundred kilobase genes are duplicated 4 times, with exonic content of >99.2% similarity[36]. This can complicate assembly because repeats result in false positive overlaps if the length of a repetitive region exceeds the length of a read. False positive overlaps then result in a cyclic or tangled graph, which is not trivially traversable, forcing the assembler to break the graph into sub-chromosomal linear chunks, or if it attempts to detangle the region, it risks creating misjoins.

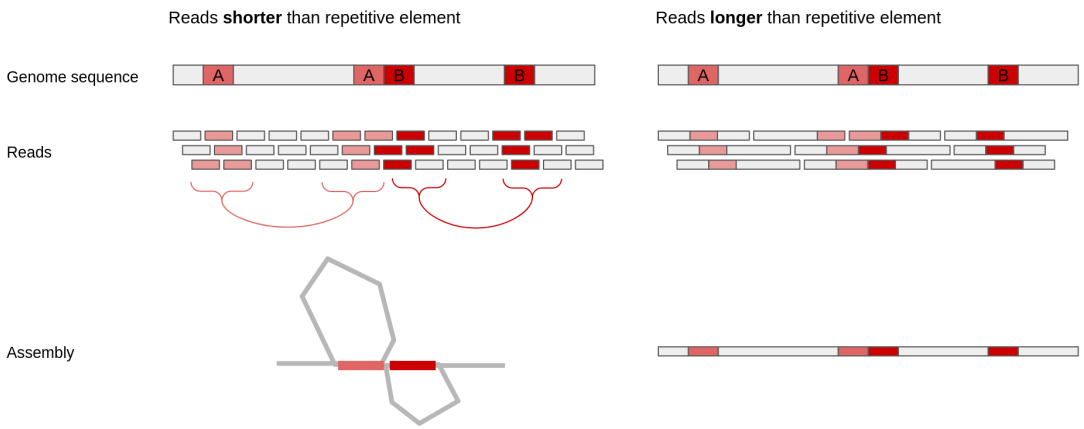

Figure 3: Diagram representation of the outcome of overlapping short (left) vs long reads (right). Repetitive regions which are inseparable at a given accuracy are shown in various shades of red. False positive overlaps are represented by red curves between short reads.

In addition to the large proportion of repetitive elements in the genome, there is a duplicated gene for every degree of ploidy that the organism has. For humans, with a ploidy of 2 and an average sequence variation rate of 0.12%[37], every chromosome has a duplicate of around 99.88% similarity. During assembly, this can either be addressed by separating the repeats with phasing, or if it is ignored, a collapsed chimeric consensus is produced. When attempting to separate haplotypes, repetitive elements which have a similarity in the same range as the haplotype similarity can complicate methods.

Variants and phasing

The vast majority of the human genome is identical, and only 0.12% is responsible for the phenotypic variation that we observe among humans[37]. Since this small fraction of sequence is useful for identifying genetic factors in disease[38], divergent sites are often isolated from the rest of the genome as “variants” or alleles, and reported in terms of minimal information needed to define an edit. Historically, genomes have been compared to a common reference genome as a way to standardize allele coordinates[37,39], but the existence of large scale variations[40] can sometimes make this comparison difficult, so alternative references which use multiple individuals are in development[17,33].

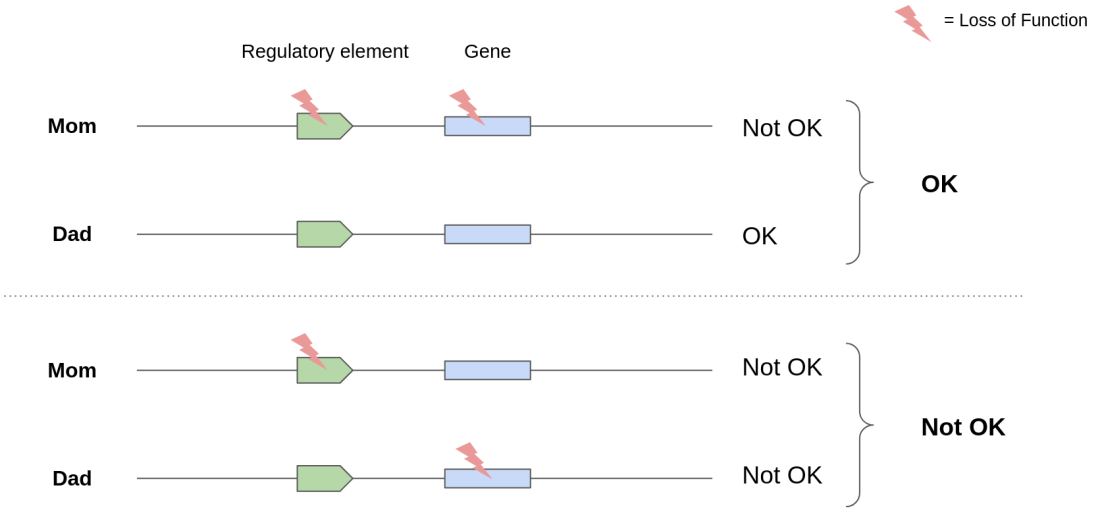

Figure 4: The combinatorial outcomes of loss of function mutations occurring in a regulon of a diploid organism. Depending on whether the loss of function occurs in one or both haplotypes, the overall outcome is either a partial or total loss, which can have drastically different phenotype or even result in viability in the case of total loss of function.

Conventional sequencing and assembly does not necessarily provide sufficient information to indicate the parental origin of an allele, which means that consecutive heterozygous variants are reported independently of one another. However, it is often important to know which mutations share a molecule because of the combinatorial outcomes of multiple variants in the same gene or regulon[41–44]. For this reason, a process called "phasing" deals with the partitioning of parental alleles into haplotypes which indicate which variants are derived from the same molecule.

Phasing generally begins with a set of heterozygous candidate variants, then the constituent alleles can be clustered by a variety of means. Given a set of pairs of alleles, long reads or linked reads inform the construction of a partition that is most consistent with the reads which span multiple variants. Consistency can be measured by the number of reads which share alleles, and therefore indicate that they come from the same molecule or chromosome[22,45,46].

Part II

Thesis

Chapter 1

Nanopore assembly and polishing

In the following work titled *Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes*, I describe methods which I developed and implemented for performing Bayesian consensus calling in nanopore homopolymers. Not described entirely in this work are several utilities I developed for assembly QC and automated cloud monitoring, which I have summarized below.

Bayesian consensus calling

Since homopolymers produce continuous signals in the nanopore sequencer, the basecaller often fails to infer the true number of bases. For this reason, a Bayesian model is used for error correction in the length domain, given a distribution of repeated samples at a locus. At the time of publication, this model was trained and deployed for inference in Marginpolish, and later was deployed as the default consensus method in Shasta. As an additional benefit of modeling homopolymers, it also became possible to evaluate homopolymer accuracy for any given sequence. This enabled the creation of figure 4b.

Cloud monitoring with TaskManager

TaskManager generates detailed CPU, RAM, and IO statistics, and sends email notifications through Amazon's Simple Email Service (SES) to ensure tasks are completed and memory or disk-related crashes are diagnosed quickly. TaskManager was used to run and monitor all software presented in this paper (except Canu), leading to figures 2f and 4d. It enabled large scale experiments such as the one in this paper, before cloud automation such as Terra existed.

Utilities for assembly QC

I created consensus accuracy QC and plotting software which was used to generate data and plots for figures 1e and 1c, showing the identity of the assembly, figure panels 2a and 2b, and all panels of figure 3.

Efficient *de novo* assembly of eleven human genomes using PromethION sequencing and a novel nanopore toolkit

Kishwar Shafin[*,1], Trevor Pesout[*,1], Ryan Lorig-Roach[*,1], Marina Haukness[*,1], Hugh E. Olsen[*,1], Colleen Bosworth[1], Joel Armstrong[1], Kristof Tigyi[1,7], Nicholas Maurer[1], Sergey Koren[4], Fritz J. Sedlazeck[5], Tobias Marschall[6], Simon Mayes[3], Vania Costa[3], Justin M. Zook[8], Kelvin J. Liu[9], Duncan Kilburn[9], Melanie Sorensen[10], Katy M. Munson[10], Mitchell R. Vollger[10], Jean Monlong[1], Erik Garrison[1], Evan E. Eichler[10,7], Sofie Salama[1,7], David Haussler[1,7], Richard E. Green[1], Mark Akeson[1], Adam Phillippy[4], Karen H. Miga[4], Paolo Carnevali[†,2], Miten Jain[†,1], and Benedict Paten[†,1]

[1] *UC Santa Cruz Genomics Institute, Santa Cruz, CA 95064, USA*
[2] *Chan Zuckerberg Initiative, Redwood City, CA 94063, USA*
[3] *Oxford Nanopore Technologies, Oxford Science Park, OX4 4DQ, UK*
[4] *Genome Informatics Section, Computational and Statistical Genomics Branch, National Human Genome Research Institute, Bethesda, MD 20892, USA*
[5] *Baylor College of Medicine, Human Genome Sequencing Center, Houston, TX 77030, USA*
[6] *Max Planck Institute for Informatics, 66123 Saarbrücken, Germany*
[7] *Howard Hughes Medical Institute, University of California, Santa Cruz, CA 95064, USA*
[8] *National Institute of Standards and Technology, Gaithersburg, MD 20899, USA*
[9] *Circulomics Inc, Baltimore, MD 21202, USA*
[10] *Department of Genome Sciences, University of Washington School of Medicine, Seattle, WA 98195, USA*
[*] *These authors contributed equally.*
[†] *Corresponding Authors.*

March 23, 2023

Abstract

Present workflows for producing human genome assemblies from long-read technologies have cost and production time bottlenecks that prohibit efficient scaling to large cohorts. We demonstrate an optimized PromethION nanopore sequencing method for eleven human genomes. The sequencing, performed on one machine in nine days, achieved an average 63x coverage, 42 Kb read N50, 90% median read identity and 6.5x coverage in 100 Kb+ reads using just three flow cells per sample. To assemble these data we introduce new computational tools: Shasta - a *de novo* long read assembler, and MarginPolish & HELEN - a suite of nanopore assembly polishing algorithms. On a single commercial compute node Shasta can produce a complete human genome assembly in under six hours, and MarginPolish & HELEN can polish the result in just over a day, achieving greater than 99.9% identity (QV30) for haploid samples from nanopore reads alone. We evaluate assembly

performance for diploid, haploid and trio-binned human samples in terms of accuracy, cost, and time and demonstrate improvements relative to current state-of-the-art methods in all areas. We further show that addition of proximity ligation (Hi-C) sequencing yields near chromosome-level scaffolds for all eleven genomes.

Introduction

Short-read sequencing reference-assembly mapping methods only assay about 90% of the current reference human genome assembly [1], and closer to 80% at high-confidence [2]. The latest incarnations of these methods are highly accurate with respect to single nucleotide variants (SNVs) and short insertions and deletions (indels) within this mappable portion of the reference genome [3]. However, short reads are much less able to *de novo* assemble a new genome [4], to discover structural variations (SVs) [5, 6] (including large indels and base-level resolved copy number variations), and are generally unable to resolve phasing relationships without exploiting transmission information or haplotype panels [7].

Third generation sequencing technologies, including linked-reads [8, 9, 10] and long-read technologies [11, 12], get around the fundamental limitations of short-read sequencing for genome inference by providing more information per sequencing observation. In addition to increasingly being used within reference guided methods [1, 13, 14, 15], long-read technologies can generate highly contiguous *de novo* genome assemblies [16].

Nanopore sequencing, as commercialized by Oxford Nanopore Technologies (ONT), is particularly applicable to *de novo* genome assembly because it can produce high yields of very long 100+ kilobase (Kb) reads [17]. 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, rDNA arrays, and recent segmental duplications [18, 19, 20]. We contributed to the recent consortium-wide effort to perform the *de novo* assembly of a nanopore sequencing based human genome [17]. This earlier effort required considerable resources, including 53 ONT MinION flow cells and an assembly process that required over 150,000 CPU hours and weeks of wall-clock time, quantities that are unfeasible for production scale replication.

Making nanopore long-read *de novo* assembly easy, cheap and fast will enable new research. It will permit both more comprehensive and unbiased assessment of human variation, and creation of highly contiguous assemblies for a wide variety of plant and animal genomes. Here we report the *de novo* assembly of eleven diverse human genomes at near chromosome scale using a combination of nanopore and proximity-ligation (HiC) sequencing [8]. We demonstrate a substantial improvement in yields and read lengths for human genome sequencing at reduced time, labor, and cost relative to earlier efforts. Coupled to this, we introduce a toolkit for nanopore data assembly and polishing that is orders of magnitude faster than state-of-the-art methods.

Results

Nanopore sequencing eleven human genomes in nine days

We selected for sequencing eleven, low-passage (six passages), human cell lines of the offspring of parent-child trios from the 1000 Genomes Project (1KGP) [21] and Genome-in-a-Bottle (GIAB) [22] sample collections. Samples were selected to maximize captured allelic diversity (see Online Methods).

We performed PromethION nanopore sequencing and HiC Illumina sequencing for the eleven

genomes. Briefly, we isolated HMW DNA from flash-frozen 50 million cell pellets using the QIAGEN Puregene kit, with some modifications to the standard protocol to ensure DNA integrity (see Online Methods). For nanopore sequencing, we performed a size selection to remove fragments <10 kilobases (Kb) using the Circulomics SRE kit, followed by library preparation using the ONT ligation kit (SQK-LSK109). We used three flow cells per genome, with each flow cell receiving a nuclease flush every 20-24 hours. 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 version 2.3.5 with the high accuracy flipflop model for basecalling (see Online Methods).

The nanopore sequencing for these eleven genomes was performed in nine days, producing 2.3 terabases of sequence. This was made possible by running up to 15 flow cells in parallel during these sequencing runs. Results are shown in Fig. 1 and Supplementary Tables **??**, **??**, and **??**. 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. 1a). The read N50s for the sequencing runs ranged between 28 Kb and 51 Kb (Fig. 1b). We aligned nanopore reads to the human reference genome (GRCh38) and calculated their alignment identity to assess sequence quality (see Online Methods). We observed that the median and modal alignment identity was 90% and 93% respectively (Fig. 1c). 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. 1d). This was in large part due to size-selection which 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 (approximately one sixth the throughput of a ProMethION flow cell) and yielded 11 Gb of data at a read N50 of 28 Kb (Supplementary Table **??**), extrapolating both are within the lower range achieved with cell line derived DNA.

Shasta: assembling a human genome from nanopore reads in under 6 hours

To assemble the genomes, we developed a new *de novo* assembly algorithm, Shasta. 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 [17]. A detailed description of algorithms and computational techniques used is provided in the Online Methods section. Here we summarize key points:

- During most Shasta assembly phases, reads are stored in a homopolymer-compressed (HPC) form using *Run-Length Encoding* (RLE) [23, 24, 25]. 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 [11]. As a result, assembly noise due to read errors is decreased, and significantly higher identity alignments are facilitated (Fig. 1e).
- 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* [26, 27] scheme is used to find candidate pairs of overlapping reads, using as *MinHash* features consecutive occurrences of m markers (default $m = 4$).

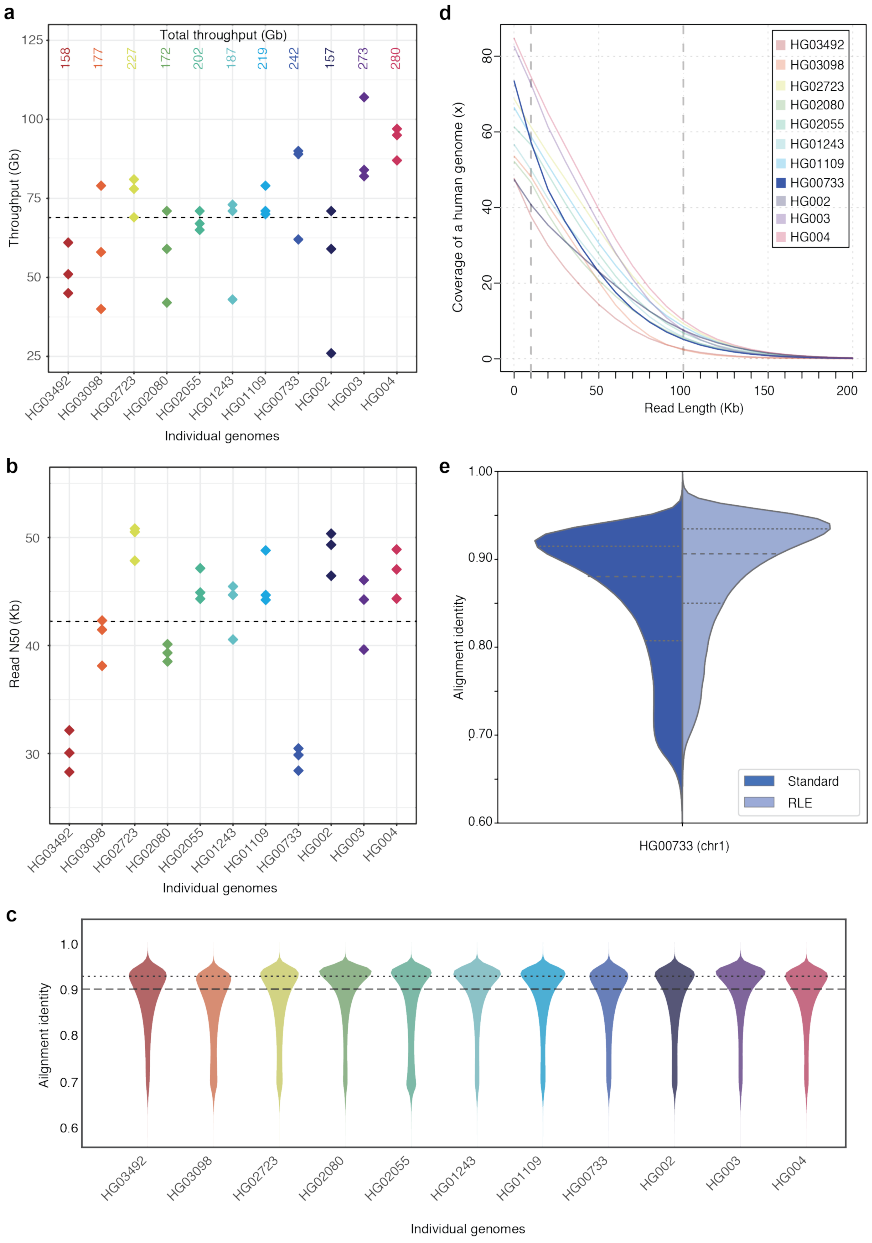

Figure 1: **Nanopore sequencing results.** **(a)** Throughput in gigabases from each of three flowcells for eleven samples, with total throughput at top. **(b)** Read N50s for each flowcell. **(c)** Alignment identities against GRCh38. Medians in a, b and c shown by dashed lines, dotted line in c is mode. **(d)** Genome coverage as a function of read length. Dashed lines indicate coverage at 10 and 100 Kb. HG00733 is bolded as an example. **(e)** Alignment identity for standard and run-length encoded (RLE) reads. Data for HG00733 chromosome 1 are shown. Dashed lines denote quartiles.

- 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.

To validate Shasta, we compared it against three contemporary assemblers: Wtdbg2 [28], Flye [29] and Canu [30]. 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 eleven samples, and data for CHM13 came from the T2T consortium [31].

Canu consistently produced the most contiguous assemblies, with contig NG50s of 40.6, 32.3, and 79.5 Mb, for samples HG00733, HG002, and CHM13, respectively (Fig. 2a). Flye was the second most contiguous, with contig NG50s of 25.2, 25.9, and 35.3 Mb, for the same samples. Shasta was next with contig NG50s of 21.1, 20.2, and 41.1 Mb. Wtdbg2 produced the least contiguous assemblies, with contig NG50s of 15.3, 13.7, and 14.0 Mb.

Conversely, aligning the samples to GRCh38 and evaluating with QUAST [32], Shasta had between 4.2 to 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 **??**). Breaking the assemblies at these disagreements and unaligned regions with respect to GRCh38, we observe much smaller absolute variation in contiguity (Fig. 2b, Supplementary Table **??**). However, a substantial fraction of the disagreements identified 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[33, 34]); in the case of HG002, we further excluded a set of known SVs [35]. We still observe between 1.2x to 2x fewer disagreements in Shasta relative to Canu and Wtdbg2, and comparable results against Flye (Fig. 2c, Supplementary Table **??**). To account for differences in the fraction of the genomes assembled, we analysed disagreements contained within the intersection of all the assemblies (i.e. in regions where all assemblers produced a unique assembled sequence). This produced results highly consistent with the prior analysis, and suggests Shasta and Flye have the lowest and comparable rates of misassembly (Online Methods, Supplementary Table. **??**). Finally, we used QUAST to calculate disagreements between the T2T Consortium's chromosome X assembly, a highly curated, validated assembly [31] and the subset of each CHM13 assembly mapping to it; Shasta has 2x to 17x fewer disagreements than the other assemblers while assembling almost the same fraction of the assembly (Supplementary Table **??**).

Canu consistently assembled the largest genomes (avg. 2.91 Gb), followed by Flye (avg. 2.83 Gb), Wtdbg2 (avg. 2.81 Gb) and Shasta (avg. 2.80 Gb). We would expect the vast majority of this assembled sequence to map to another human genome. Discounting unmapped sequence, the differences are smaller: Canu produced an avg. 2.86 Gb of mapped sequence per assembly, followed by Shasta (avg. 2.79 Gb), Flye (avg. 2.78 Gb) and Wtdbg2 (avg. 2.76 Gb) (Fig. 2d; see Online Methods). This analysis supports the notion that Shasta is currently relatively conservative vs. its peers, producing the highest proportion of directly mapped assembly per sample.

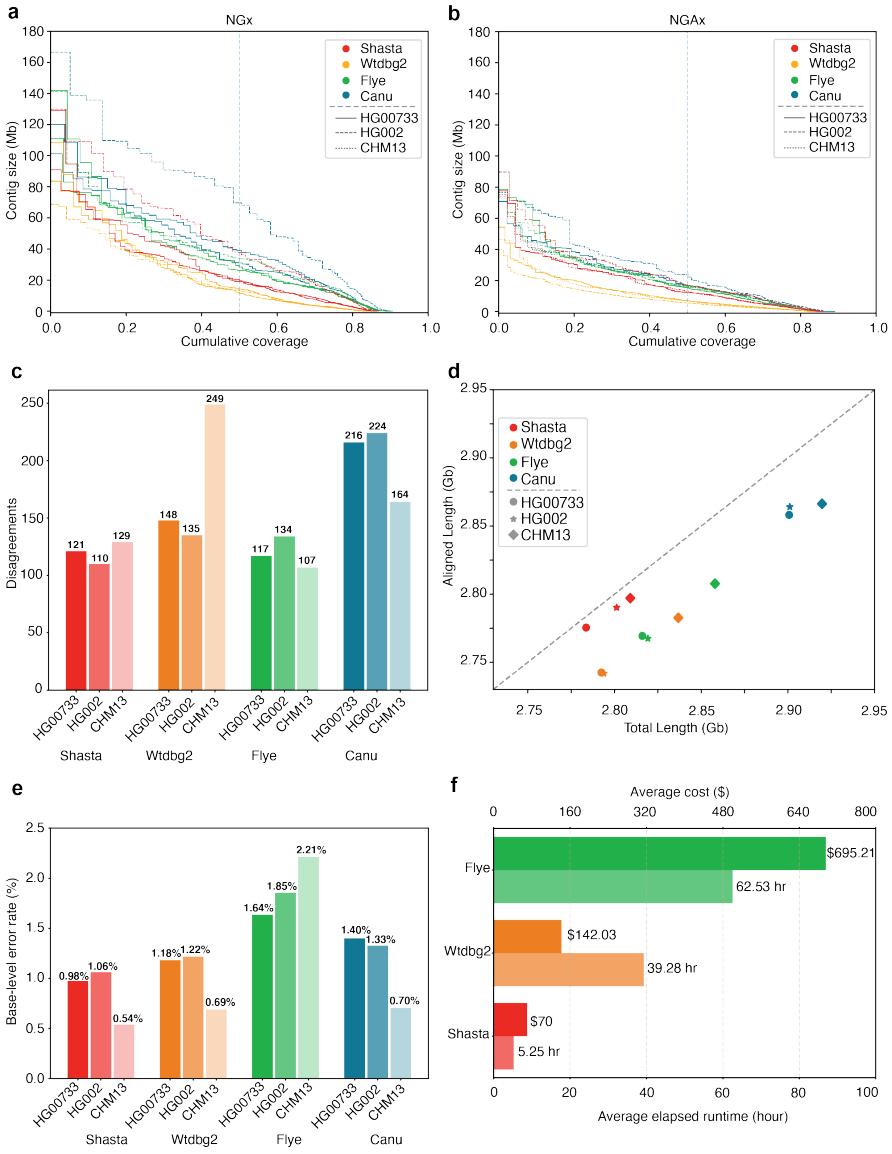

Figure 2: **Assembly results for four assemblers and three human samples, before polishing. (a)** NGx plot showing contig length distribution. The intersection of each line with the dashed line is the NG50 for that assembly. **(b)** NGAx plot showing the distribution of *aligned* contig lengths. Each 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 of centromeres, segmental duplications and, for HG002, known SVs. **(d)** Total generated sequence length vs. 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).

For HG00733 and CHM13 we examined a library of bacterial artificial chromosome (BAC) assemblies (see Online 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 Online Methods), we find Shasta contiguously assembles all 47 BACs, with Flye performing similarly (Supplementary Table **??**). 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 **??**). 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 vs. its peers (Canu: 92%, Flye 87%, Wtdbg2 88%). In the intersected set of BACs attempted by all assemblers (Supplementary Table **??**) Shasta: 100%, Flye: 100%, Canu: 98.50% and Wtdbg2: 90.80% all produce comparable results.

Shasta produced the most base-level accurate assemblies (avg. 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. 2e); see Online Methods, Supplementary Table **??**. 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 **??**).

Shasta, Wtdbg2 and Flye were run on a commercial cloud, allowing us to reasonably compare their cost and run time (Fig. 2e; see Online Methods). Shasta took an average of 5.25 hours to complete each assembly at an average cost of $70 per sample. In contrast, Wtdbg2 took 7.5x longer and cost 3.7x as much, and Flye took 11.9x longer and cost 9.9x as much. The Canu assemblies were run on a large compute cluster, consuming up to $19,000 (estimated) of compute and took around 4-5 days per assembly (see Online Methods, Supplementary Tables **??**, **??**).

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

Contiguously assembling MHC haplotypes

The Major Histocompatibility Complex (MHC) region is difficult to resolve using short reads due to its repetitive and highly polymorphic nature [36], but recent efforts to apply long read sequencing to this problem have shown promise [17, 37]. We analyzed the assemblies of CHM13 and HG00733 to see if they spanned the region. For the haploid assembly of CHM13 we find MHC is entirely spanned by a single contig in all 4 assemblers' output, and most closely resembles the GL000251.2 haplogroup among those provided in GRCh38 (Fig. 3a; Supplementary Fig. **??** and Supplementary Table **??**). In the diploid assembly of HG00733 two contigs span the large majority of the MHC for Shasta and Flye, while Canu and Wtdbg2 span the region with one contig (Fig. 3b; Supplementary Fig. **??**). However, we note that the chimeric diploid assembly leads to sequences that do not closely resemble any haplogroup (see Online Methods).

To attempt to resolve haplotypes of HG00733 we performed trio-binning [38], where we partitioned all the reads for HG00733 into two sets based on likely maternal or paternal lineage and assembled the haplotypes (see Online Methods). For all haplotype assemblies the global contiguity worsened significantly (as the available read data coverage was approximately halved, and further, not all reads could be partitioned), but the resulting disagreement count decreased (Supplementary

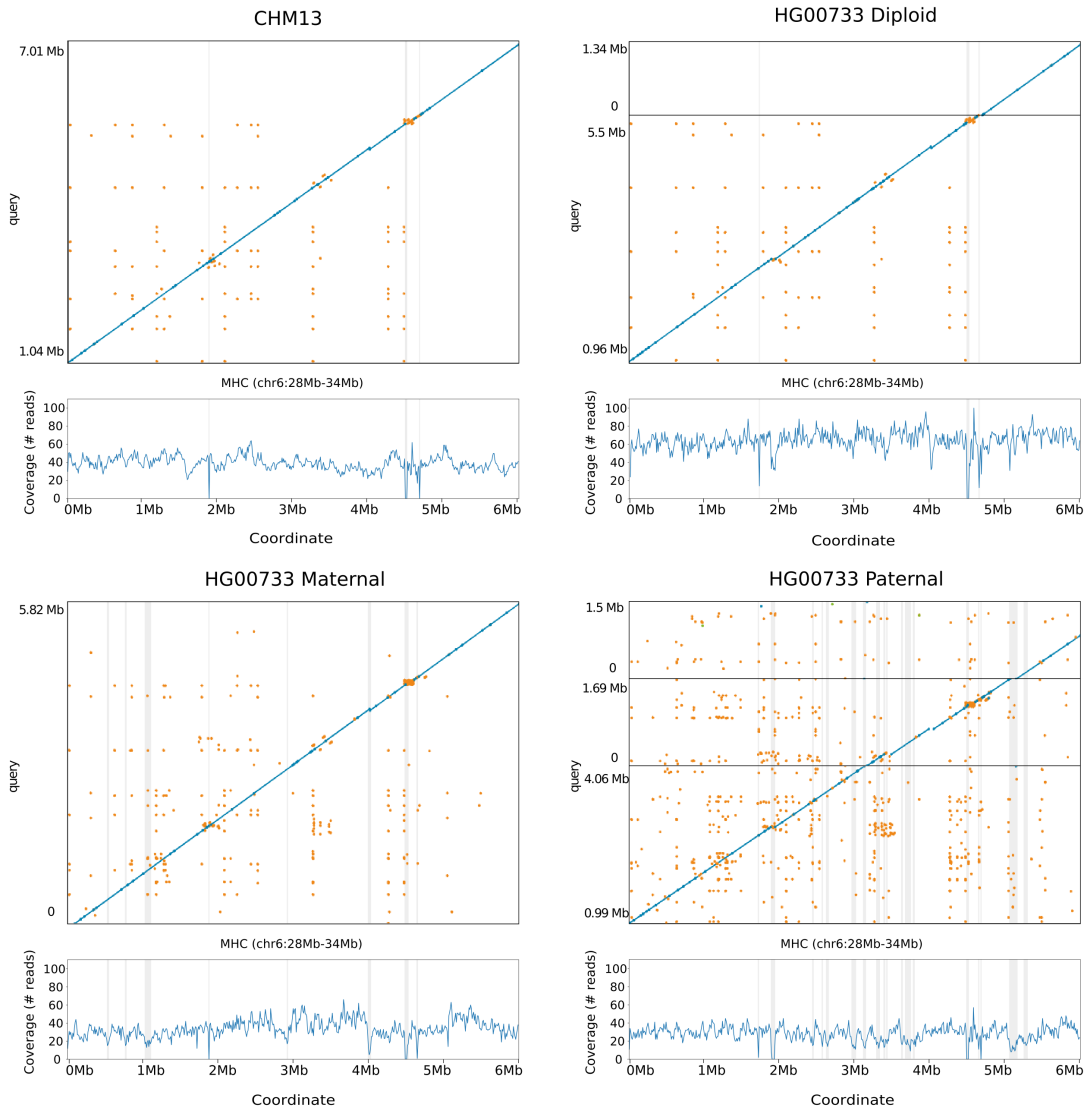

Figure 3: **Shasta MHC assemblies vs GRCh38.** Unpolished Shasta assembly for CHM13 and HG00733, including HG00733 trio-binned maternal and paternal assemblies. Shaded gray areas are regions in which coverage (as aligned to GRCh38) drops below 20. Horizontal black lines indicate contig breaks. Blue and green describe unique alignments (aligning forward and reverse, respectively) and orange describes multiple alignments.

Table **??**). When using haploid trio-binned assemblies, the MHC was spanned by a single contig for the maternal haplotype (Fig. 3c, Supplementary Fig. **??**, Supplementary Table **??**), 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. 3d) breaking the region into three contigs.

Deep neural network based polishing achieves greater than QV30 long-read only haploid polishing accuracy

Accompanying Shasta, 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 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 [39]. From these statistics MarginPolish generates a weighted RLE Partial Order Alignment (POA) graph [40] 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 employs a multi-task recurrent neural network (RNN) [41] 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 [42] followed by Medaka [43]. Here 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 4a and Supplementary Tables **??**, **??** and **??** detail error rates for the four methods performed on the HG00733 and CHM13 Shasta assemblies (see Online Methods) using Pomoxis [44]. For the diploid HG00733 sample MarginPolish and HELEN achieve a balanced error rate of 0.388% (QV 24.12), compared to 0.455% (QV 23.42) by Racon and Medaka. For both polishing pipelines, a significant 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 [31]. We achieve a balanced error rate of 0.064% (QV 31.92), compared to Racon and Medaka's 0.110% (QV 29.59).

For all assemblies, errors were dominated by indel errors, e.g. substitution errors are 3.16x and 2.9x fewer than indels in the polished HG000733 and CHM13 assemblies, respectively. Many of these errors relate to homopolymer length confusion; Fig. 4b 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 Online Methods, Supplementary Fig. **??**).

Figure 4c and Supplementary Table **??** 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 [45]. This led to a ~2x reduction in base errors, increase the Q score from ~QV32 (after polishing with MarginPolish and HELEN) to ~QV36 (Supplementary Table **??**). Notably, attempting to use Pilon polishing on the raw Shasta assembly resulted in much poorer results (QV24).

Figure 4d and Supplementary Table **??** describe average runtimes and costs for the methods (see Online Methods). MarginPolish and HELEN cost a combined $107 and took 29 hours of wall-clock time on average, per sample. In comparison Racon and Medaka cost $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 3.0x faster than Racon (1x)/Medaka (Supplementary Table **??**).

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 [46], which can annotate a genome assembly using the human GENCODE [47] reference human gene set (Table 1, Online Methods, Supplementary Tables **??**, **??**, **??**, and **??**.).

Table 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 were close to representing 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 [48] using the eukaryote set of orthologs on each assembly, a smaller

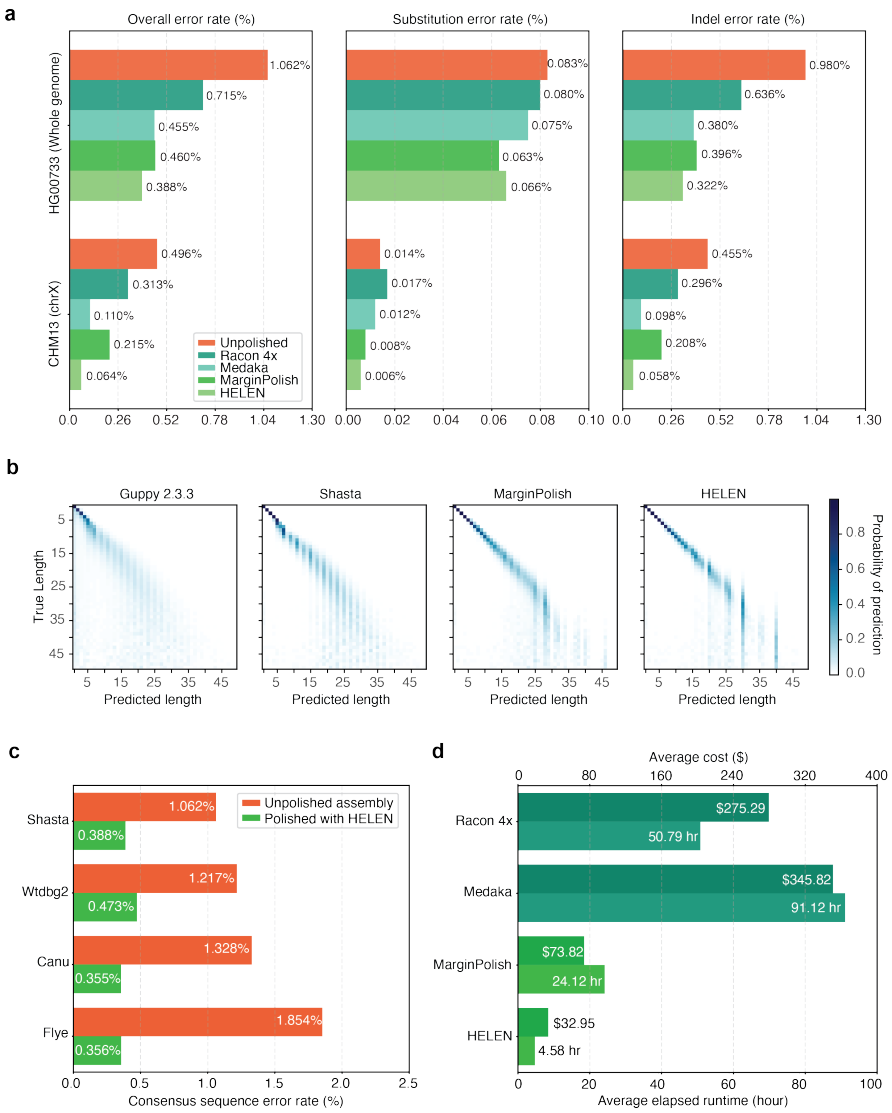

Figure 4: **Polishing Results. (a)** Balanced error rates for the four methods on HG00733 and CHM13. **(b)** Row-normalized heatmaps describing the predicted run-lengths (x-axis) given true run lengths (y-axis) for four steps of the pipeline on HG00733. **(c)** Error rates for MarginPolish and HELEN on four assemblies. **(d)** Average runtime and cost.

set of 303 expected single-copy genes (Supplementary Tables **??** and **??**). We find comparable performance between the assemblies, with small differences largely recapitulating the pattern observed by the larger CAT analysis.

Comparing to a PacBio HiFi Assembly

We compared the CHM13 Shasta assembly polished using MarginPolish and HELEN with the recently released Canu assembly of CHM13 using PacBio HiFi reads [49]; HiFi reads being based upon circular consensus sequencing technology that delivers significantly lower error rates. The HiFi assembly has lower NG50 (29.0 Mb vs. 41.0 Mb) than the Shasta assembly (Supplementary Fig. **??**). 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 (8469), a difference which remains after looking only at disagreements within the intersection of the assemblies (380 vs. 594). The assemblies have an almost equal NGAx (~20.0Mb), but the Shasta assembly covers a smaller fraction of GRCh38 (95.28% vs. 97.03%) (Supplementary Fig. **??**, Supplementary Table **??**). Predictably, the HiFi assembly has a higher QV value than the polished Shasta assembly (QV41 vs. QV32).

Assembling, polishing and scaffolding 11 human genomes at near chromosome scale

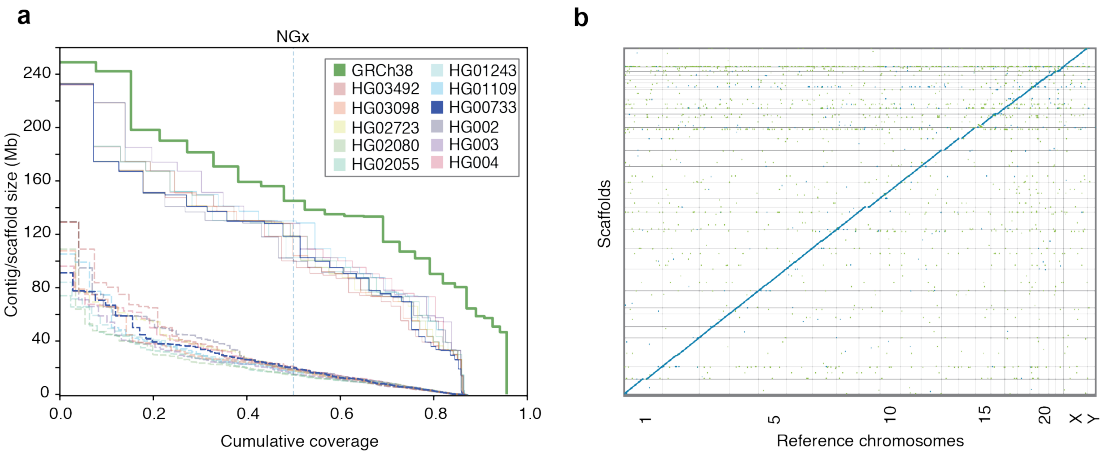

Figure 5: **HiRise scaffolding for 11 genomes. (a)** NGx plots for each of the 11 genomes, before (dashed) and after (solid) scaffolding with HiC sequencing reads, GRCh38 minus alternate sequences is shown for comparison. **(b)** Dot plot showing alignments between the scaffolded HG00733 Shasta assembly and GRCh38 chromosome scaffolds. Blue indicates forward aligning segments, green indicates reverse, with both indicating unique alignments.

To achieve chromosome length sequences we scaffolded all of the polished Shasta assemblies with HiC proximity-ligation data using HiRise [50] (see Online Methods, Fig. 5a). On average, 891 joins were made per assembly. This increased the scaffold NG50s to near chromosome scale, with a median of 129.96 Mb, as shown in Fig. 5a, with additional assembly metrics in Supplementary Table **??**. 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 major rearrangements and all chromosomes are spanned by one or a few contigs (Fig. 5b), with the exception of chrY which is

absent because HG00733 is female. Similar results were observed for HG002 (Supplementary Fig. **??**).

Discussion

In this paper we demonstrate the sequencing and assembly of eleven diverse human genomes in a time and cost efficient manner using a combination of nanopore and proximity ligation sequencing.

The PromethION realizes dramatic improvements in yield per flow cell, allowing the sequencing of each genome with just three flow cells at an average coverage of 63x. This represents a large reduction in associated manual effort and a dramatic practical improvement in parallelism; a single PromethION allows up to 48 flow cells to be run concurrently. Here we completed all 2.3 terabases of nanopore data collection in nine days on one PromethION, running up to 15 flow cells simultaneously (it is now possible to run 48 concurrently). In terms of contemporary long-read sequencing platforms, this throughput is unmatched.

Due to the length distribution of human transposable elements, we found it better to discard reads shorter than 10 Kb to prevent multi-mapping. The Circulomics SRE kit reduced the fraction of reads <10 Kb to around 13%, making the majority usable for assembly. Conversely, the right tail of the read length distribution is long, yielding an average of 6.5x coverage per genome in 100 Kb+ reads. This represents an enrichment of around 7 fold relative to our earlier MinION effort [17]. In terms of assembly, the result was an average NG50 of 18.5 Mb for the 11 genomes, ~3x higher than in that initial effort, and comparable with the best achieved by alternative technologies [12, 51]. We found the addition of HiC sequencing for scaffolding necessary to achieve chromosome scale, making 891 joins on average per assembly. However, our results are consistent with previous modelling based on the size and distribution of large repeats in the human genome, which predicts that an assembly based on 30x coverage of such 100 Kb+ reads would approach the continuity of complete human chromosomes [17, 31].

Relative to alternate long-read and linked-read sequencing, the read identity of nanopore reads has proven lower [11, 17]. However, original reports of 66% identity [11] for the original MinION are now historical footnotes: we observe modal read identity of 92.5%, resulting in better than QV30 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 reads [51] and 10x Chromium [52], will likely prove useful in achieving QV40+ assemblies.

The advent of third generation technologies has dramatically lowered the cost of high-contiguity long-read *de novo* assembly relative to earlier methods [53]. This cost reduction is still clearly underway. The first MinION human assembly cost ~$40,000 in flow cells and reagents [17]. After a little over a year, the equivalent cost per sample here was ~$6,000. At bulk with current list-pricing, this cost would be reduced to ~$3,500 per genome. It is not unreasonable to expect further yield growth and resulting cost reduction of nanopore and competing platforms such that we foresee $1,000 total sequencing cost high-contiguity *de novo* plant and animal genome assembly being achieved - a milestone that will likely make many ambitious comparative genomic efforts economic [54, 55].

With sequencing efficiency for long-reads improving, computational considerations are paramount in figuring 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 novel methods that provide a pipeline for the rapid assembly of long nanopore

reads. Shasta can produce a draft human assembly in around six hours and $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 7.5x slower and 3.7x more expensive. Connected together, the three tools presented allow a polished assembly to be produced in ~24 hours and for ~$180, against the fastest comparable combination of Wtdbg2, Racon, and Medaka which costs 5.3x more and is 4.3x slower while producing measurably worse results in terms of disagreements, contiguity and base-level accuracy. Substantial further parallelism of polishing, the dominant time component in our current pipeline, is easily possible. We are now working toward the goal of having a half-day turn around of our complete computational pipeline. With real-time base calling, a DNA-to-*de novo* assembly could be achieved in less than 96 hours with little difficulty. Such speed could make these techniques practical for screening human genomes for abnormalities in difficult-to-sequence regions.

All three presented computational methods employ run-length encoding of reads. By operating on homopolymer-compressed nucleotide sequences, we mitigate effects of the dominant source of error in nanopore reads [56] and enable the use of different models for addressing alignment and run-length estimation orthogonally.

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 4x 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 have 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 [16, 57, 58]. We anticipate that the novel tools we've 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 reads [1], and the HELEN model could be improved to include haplotagged features for the identification of heterozygous sites.

Acknowledgements

The authors are grateful for support from the following individuals. Dan Turner, David Stoddart, Androo Markham, and Jonathan Pugh (ONT) provided advice on method development and basecalling. Chris Wright (ONT) provided advice for Medaka. Daniel Garalde, and Rosemary Dokos (ONT) provided advice on the PromethION for parallelized DNA sequencing and basecalling.

The authors are grateful to Amazon Web Services (AWS) for hosting the data via their AWS Public Dataset Program.

AP and SK were supported by the Intramural Research Program of the National Human Genome Research Institute, National Institutes of Health. This work utilized the computational resources of the NIH HPC Biowulf cluster (`https://hpc.nih.gov`).

Sidney Bell and Charlotte Weaver from Chan Zuckerberg Initiative (CZI) provided support on development and documentation. CZI further supported this effort by funding the usage of Amazon Web Services (AWS) for the project.

Certain commercial equipment, instruments, or materials are identified to specify adequately experimental conditions or reported results. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment, instruments, or materials identified are necessarily the best available for the purpose.

This work was supported, in part, by the National Institutes of Health (award numbers: 5U54HG007990, 5T32HG008345-04, 1U01HL137183, R01HG010053, U01HL137183, and U54HG007990 to BP and DH; R01HG010329 to SRS and DH), by Oxford Nanopore Research Grant SC20130149 (MA), the Howard Hughes Medical Institute (DH), the Gencode (NIH/NHGRI/EMBL) Grant 2U41HG007234, TopMEd (NIH/NHLBI) Grant 1U01HL137183, W.M. Keck Foundation Grant DT06172015, BD2K Grant 1U54HG007990, the technological pilot using ONT nanopore sequencing (NIH/NHGRI) Grant 3U24HG009084-03S1, and High coverage long read nanopore sequencing of reference human genomes (NIST) Grant 70NANB18H224.

Code Availability

The three novel software tools, Shasta (`https://github.com/chanzuckerberg/shasta`), MarginPolish (`https://github.com/UCSC-nanopore-cgl/marginPolish`), and HELEN (`https://github.com/kishwarshafin/helen`) are publicly available. They have open source MIT license which fully supports the open source initiative.

Online Methods

Sample selection

The goal of sample selection was to select a set of individuals that collectively captured the maximum amount of weighted allelic diversity [59]. 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 [60] (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 [22].

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°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. 1c/e) we aligned all the reads for each sample and flowcell to GRCh38 using `minimap2` [23] 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 1c and Figure 1e. For Figure 1e, 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/`.

Table 2: The truth assembly files with download URLs.

Sample name	Region	File type	URL
HG002	Whole genome	fasta	`HG002_GRCh38_h1.fa`
		bed	`HG002_GRCh38.bed`
HG00733	Whole genome	fasta	`hg00733_truth_assembly.fa`
CHM13	Whole genome	fasta	`CHM13_truth_assembly.fa`
	Chr-X	fasta	`CHRX_CHM13_truth_assembly.fa`

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 [61]. 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 [31]. 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 $>= 40$) we have.

QUAST / BUSCO

To quantify contiguity, we primarily depended on the tool QUAST [32]. 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 2d. 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 **??**. 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 2b) 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 [48] 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 [7], centromeric regions for GRCh38, and known regions in GRCh38 with structural variation for HG002 from GIAB [35]. 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 defined in the `ucsc.collapsed.sorted.segdups` file 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 available in `NIST_SVs_Integration_v0.6/` under `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 here `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 -end-bonus=100 -O 5,56 -E 4,1 -B 5`. 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 [38]. 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 [22]. 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 [46] 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 [62] to create the input alignment to CAT. The GENCODE [63] 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. 2f 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.

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 [64] 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 [65] 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 [66, 67] and 179 for HG00733 [7] (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 >= 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 [45] 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/gimbricate c1c6d1a`) 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/vg v1.19.0 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, [68]). 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 (`https://github.com/chanzuckerberg/shasta/releases/tag/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 [23, 24, 25].

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

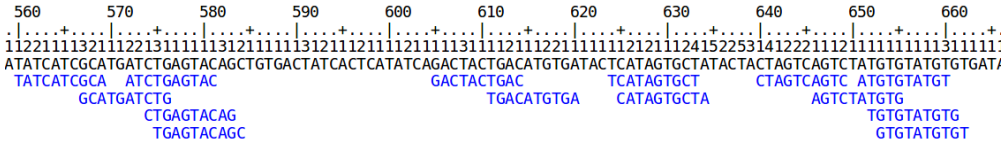

Figure 6: Markers aligned to a run length encoded read.

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 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. 7 shows an example alignment matrix.

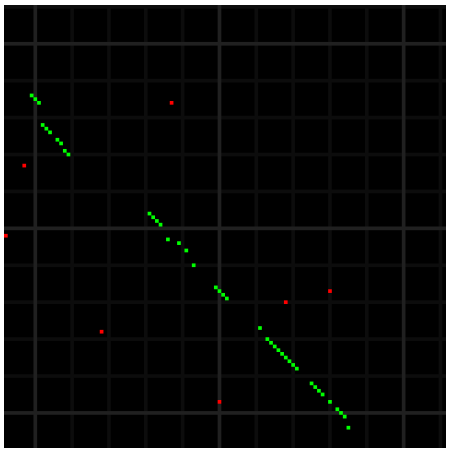

Figure 7: A marker alignment represented as a dot-plot. Elements that are identical between the two sequences are displayed in green or red - the ones in green are the ones that are part of the optimal alignment computed by the Shasta assembler. Because of the much larger alphabet, matrix elements that 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.

Computing optimal alignments in marker representation

To compute the (likely) optimal alignment (example highlighted in green in Fig. 7), 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. 7, 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 (∼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 [26, 27] 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 Jaccard 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 [69]. 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 occurrences 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 (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 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.

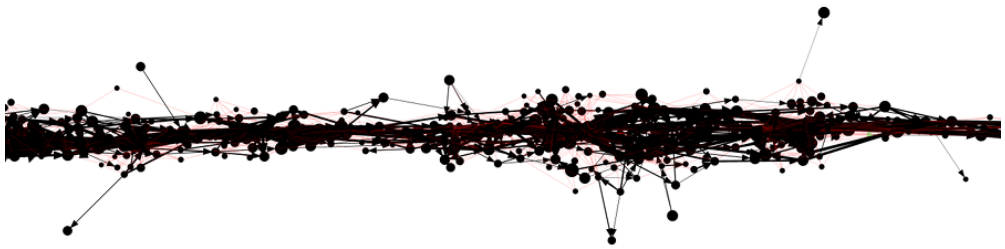

Figure 8: An example of a portion of the read graph, as displayed by the Shasta http server.

An example of one strand is shown in Fig. 8. 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 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. 9), typically for high similarity segment duplications.

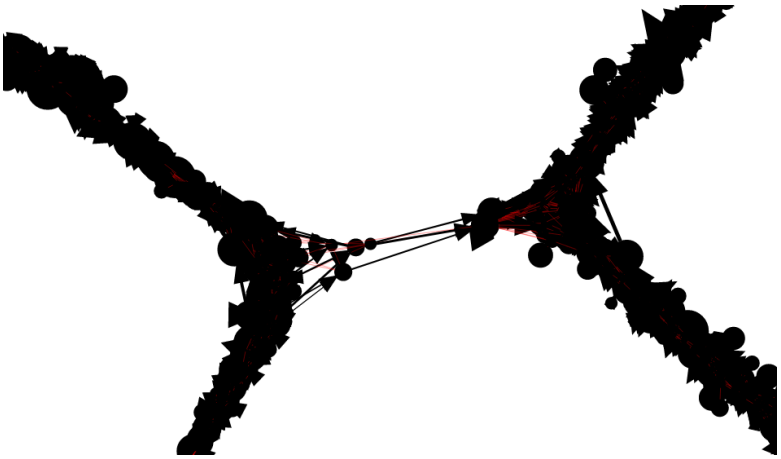

Figure 9: An example of a portion of the read graph showing obviously incorrect connections

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.

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. 10).

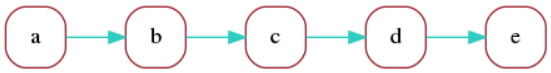

Figure 10: A marker graph representing a single read.

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
```

The marker graph for the two reads is Fig 11(A).

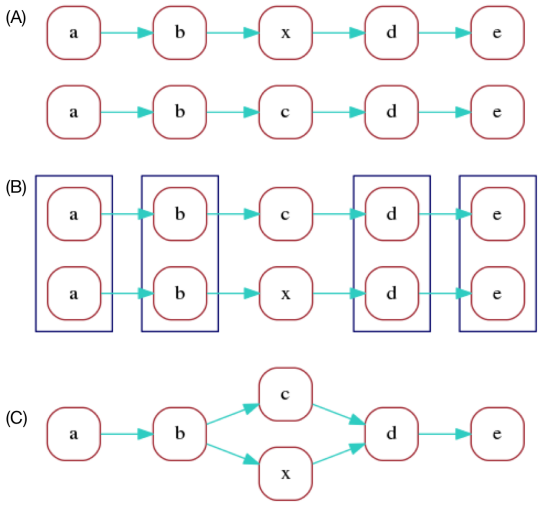

Figure 11: An illustration of marker graph construction for two sequences.

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 11(B).

Finally, we can merge aligned vertices to obtain a marker graph describing the two aligned reads, shown in Fig 11(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.

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 [70], 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. 12).

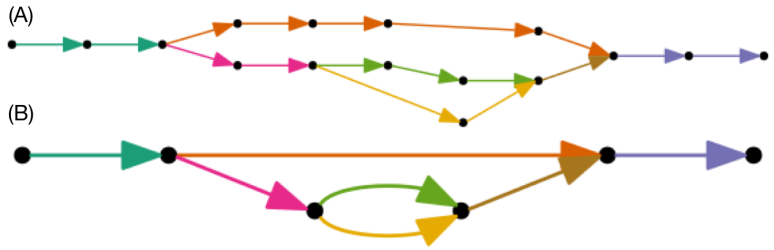

Figure 12: (A) A marker graph with linear sequence of edges colored. (B) The corresponding assembly graph. Colors were chosen to indicate the correspondence to marker graph edges.

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 [40] (`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:

- Approximate transitive reduction of the marker graph.
- Pruning of short side branches (leaves).
- 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.

Approximate transitive reduction of the marker graph

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 Breadth-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 superbubbles, which are handled in the next two phases described below.

Pruning of short side branches (leaves)

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).

Removal of bubbles and superbubbles

The marker graph now consists of mostly linear section with occasional bubbles or superbubbles [71]. 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 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. 13 shows typical examples of a bubble and superbubble in the marker graph.

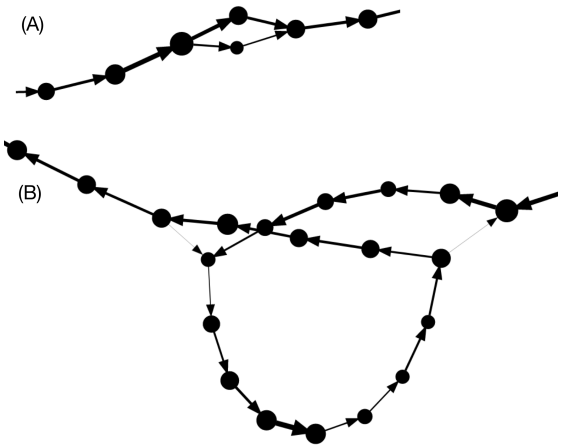

Figure 13: (A) A simple bubble. (B) A superbubble.

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 iterarion. 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 assembler 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 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, 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 for conveniently handling these large memory-mapped regions as C++ containers with familiar semantics (`class shasta::MemoryMapped::Vector`).
- 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 [40] (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 [42], 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 [1], 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 [72]. Most earlier algorithms for constructing POA graphs have a well known explicit read order dependence that can result in undesirable topologies [40].
- MarginPolish works in run-length encoded space, which results in considerably less alignment uncertainty and correspondingly improved performance.
- MarginPolish, similarly to Nanopolish [73], 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 [62]. 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 [11] of R9.4 ONT reads aligned to a bacterial reference; we have observed the parameters for this HMM seem robust to small changes in base-caller 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 [74]. 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.

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. 14(A) shows an example).

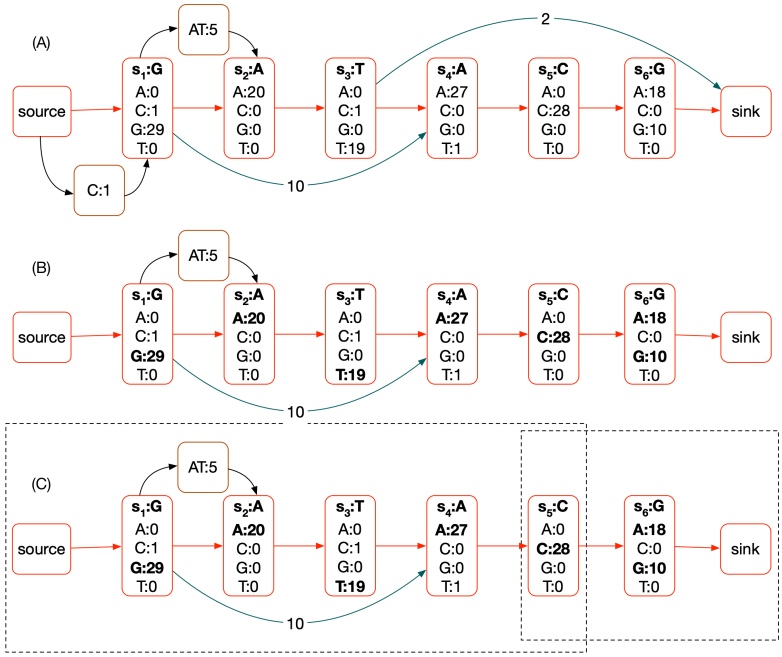

Figure 14: A) An example POA, assuming approximately 30x read coverage. The backbone is shown in red. Each non-source/sink node has a vector of weights, one for each possible base. Deletion edges are shown in teal, they also each have a weight. Finally insertion nodes are shown in brown, each also has 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.

Frequently either an insertion or deletion can be made between different successive 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. 14 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. 14(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 separated 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

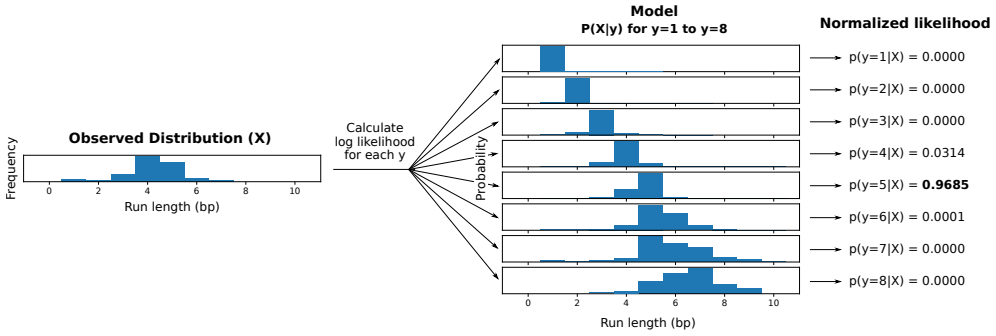

Figure 15: **Visual representation of run length inference.** This diagram shows how a consensus run length is inferred for a set of aligned lengths (X) that pertain to a single position. The lengths are factored and then iterated over, and log likelihood is calculated for every possible true length 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.

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. 15).

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. 18).

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) [41] 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.

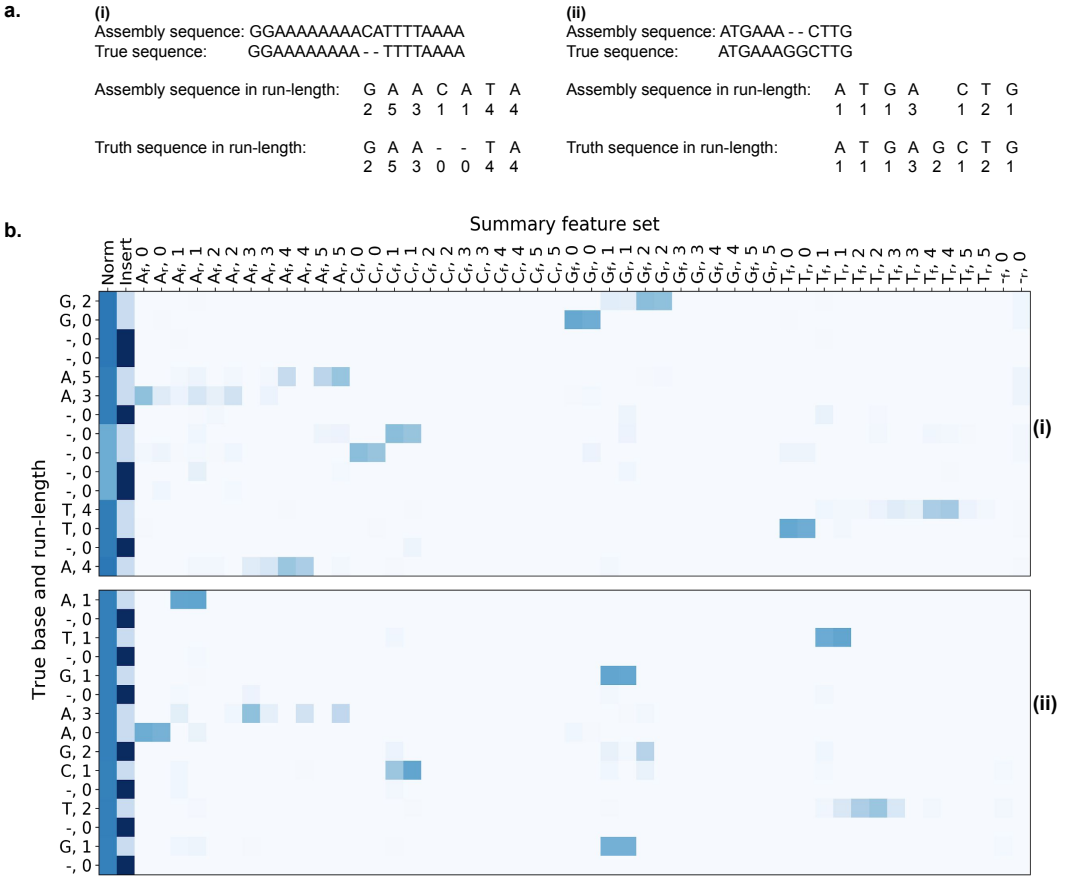

Figure 16: **MarginPolish Images** A graphical representation of images from two labeled regions selected to demonstrate: the encoding of a single POA node into two run-length blocks (i), a true deletion (i), and a true insert (ii). The y-axis shows truth labels for nucleotides and run-lengths, the x-axis describes features in the images, and colors show associated weights.

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, 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. 16 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. 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.

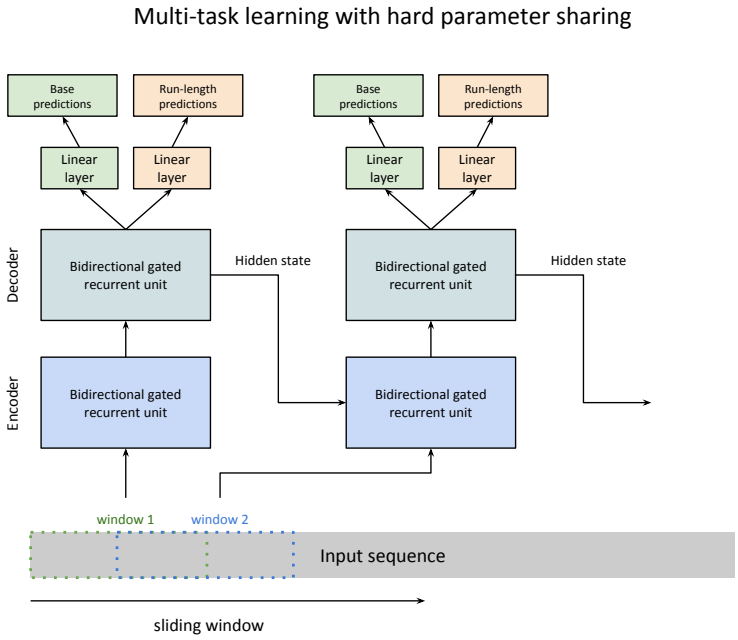

Figure 17: The sequence-to-sequence model implemented in Helen.

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:

$$
\begin{aligned}
r_t &= Sigmoid(W_{ir}x_t + W_{hr}h_{(t-1)}) \\
u_t &= Sigmoid(W_{iu}x_t + W_{hu}h_{(t-1)}) \\
n_t &= tanh(W_{in}x_t + r_t * (W_{hn}h_{(t-1)})) \\
h_t &= (1 - u_t) * n_t + u_t * h_{(t-1)}
\end{aligned}
\tag{1}
$$

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:

$$
\begin{aligned}
B_t &= h_t * W^T \\
R_t &= h_t * W^T
\end{aligned}
\tag{2}
$$

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 2 using the same set of underlying parameters from equation 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 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[41]. 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 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:

$$\begin{aligned} L_B(B, T_B) &= -B[T_B] + \log\left(\textstyle\sum_j \exp(B[j])\right) \\ L_R(R, T_R) &= weight[T_R]\left(-R[T_R] + \log\left(\textstyle\sum_j \exp(R[j])\right)\right) \\ L &= L_B + L_R \end{aligned} \tag{3}$$

In equation 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 1 and equation 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 **??**, **??** and **??** 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 3: 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 Chr20 as the final model.

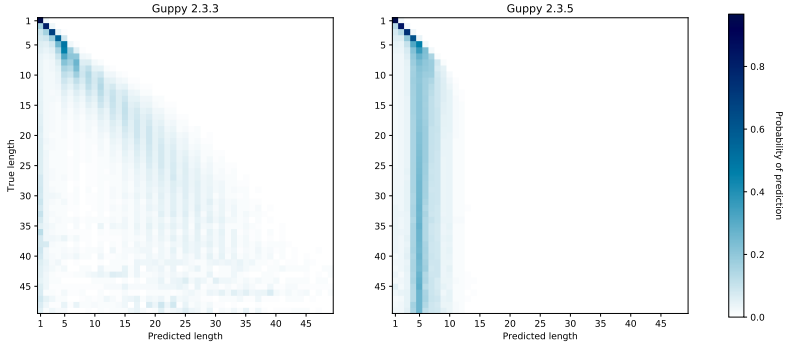

Figure 18: Run-length confusion in different versions of Guppy base caller

The CHM13 data from T2T consortium [31] 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 18 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 [75] deep neural network library for the model implementation. We also use the Striped-Smith Waterman algorithm implementation to use during stitching and Pybind11 [76] 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`).

References

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

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56, 2012.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

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

[42] 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.

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

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] Zhanshan Sam Ma, Lianwei Li, Chengxi Ye, Minsheng Peng, and Ya-Ping Zhang. Hybrid assembly of ultra-long nanopore reads augmented with 10x-genomics contigs: Demonstrated with a human genome. *Genomics*, 2018.

[53] Hayan Lee, James Gurtowski, Shinjae Yoo, Maria Nattestad, Shoshana Marcus, Sara Goodwin, W Richard McCombie, and Michael Schatz. Third-generation sequencing and the future of genomics. *BioRxiv*, page 048603, 2016.

[54] Genome 10K Community of Scientists. Genome 10k: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

[55] Harris A Lewin, Gene E Robinson, W John Kress, William J Baker, Jonathan Coddington, Keith A Crandall, Richard Durbin, Scott V Edwards, Félix Forest, M Thomas P Gilbert,

et al. Earth biogenome project: Sequencing life for the future of life. *Proceedings of the National Academy of Sciences*, 115(17):4325–4333, 2018.

[56] Franka J Rang, Wigard P Kloosterman, and Jeroen de Ridder. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome biology*, 19(1):90, 2018.

[57] 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.

[58] 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.

[59] 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.

[60] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

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

[62] 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.

[63] 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.

[64] 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.

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

[66] 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.

[67] 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.

[68] 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, Sayed Mohammad E. Sahraeian, Vincent Huang, Alexandre Rouette, Noah Alexander, Christopher E. Mason, Iman Hajirasouliha, Camir Ricketts, Joyce Lee, Rick Tearle, Ian T. Fiddes, Alvaro Martinez Barrio, Jeremiah Wala, Andrew Carroll, Noushin Ghaffari, Oscar L. Rodriguez, Ali Bashir, Shaun Jackman, John J Farrell, Aaron M Wenger, Can Alkan, Arda Soylev, Michael C. Schatz, Shilpa Garg, George Church, Tobias Marschall, Ken Chen, Xian Fan, Adam C. English, Jeffrey A. Rosenfeld, Weichen Zhou, Ryan E. Mills, Jay M. Sage, Jennifer R. Davis, Michael D. Kaiser, John S. Oliver, Anthony P. Catalano, Mark JP

Chaisson, Noah Spies, Fritz J. Sedlazeck, Marc Salit, and . A robust benchmark for germline structural variant detection. *bioRxiv*, 2019.

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

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

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

[72] 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.

[73] 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.

[74] 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.

[75] 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.

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

Chapter 2

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

In the following work titled *Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with GetBlunted*, I developed the core method of aligning and duplicating sequences using POA to generate blunt ended representations of sequences. I implemented and tested all aspects of this work except for the algorithms for computing the biclique cover of overlaps.

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

Jordan M. Eizenga*[0000−0001−8345−8356], Ryan Lorig-Roach*[0000−0002−8183−9611], Melissa M. Meredith[0000−0001−5736−3193], and Benedict Paten[0000−0001−8863−3539]

University of California Santa Cruz Genomics Institute
1156 High Street, Santa Cruz, CA 95064
bpaten@ucsc.edu

* Contributed equally

Abstract. Sequence graphs have emerged as an important tool in two distinct areas of computational genomics: genome assembly and pangenomics. However, despite this shared basis, subtly different graph formalisms have hindered the flow of methodological advances from pangenomics into genome assembly. In genome assembly, edges typically indicate overlaps between sequences, with the overlapping sequence expressed redundantly on both nodes. In pangenomics, edges indicate adjacency between sequences with no overlap—often called *blunt* adjacencies. Algorithms and software developed for blunt sequence graphs often do not generalize to overlapped sequence graphs. This effectively silos pangenomics methods that could otherwise benefit genome assembly. In this paper, we attempt to dismantle this silo. We have developed an algorithm that transforms an overlapped sequence graph into a blunt sequence graph that preserves walks from the original graph. Moreover, the algorithm accomplishes this while also eliminating most of the redundant representation of sequence in the overlap graph. The algorithm is available as a software tool, GetBlunted, which uses little enough time and memory to virtually guarantee that it will not be a bottleneck in any genome assembly pipeline.

Keywords: Genome assembly · Graph genome · Pangenomics.

1 Introduction

Genome assembly is the process of determining a sample's full genome sequence from the error-prone, fragmentary sequences produced by DNA sequencing technologies. Sequence graphs have a long history of use in this field [16, 20, 17]. In these graphs, nodes are labeled with sequences derived from sequencing data, and edges indicate overlaps between observed sequences, which may in turn indicate adjacency in the sample's genome (Fig. 1A). The sample genome then

corresponds to some walk through graph. There are several specific sequence graph articulations in wide use, including de Bruijn graphs, overlap graphs, and string graphs. They each present computational and informational trade-offs that make them better suited to certain configurations of sequencing technologies and genome complexity.

The common topological features of genome assembly graphs are driven primarily by the repetitiveness of the underlying genomes. In many species, a large fraction of the genome consists of repeats (for instance, more than 50% of the human genome [11]). Because all copies of a repeat are highly similar to each other, the corresponding nodes in the sequence graph frequently overlap each other. In contrast, the unique regions of the genome have few erroneous overlaps. These two factors tend to create graphs that consist of long non-branching paths (corresponding to the unique regions), which meet in a densely tangled core with a complicated topology (corresponding to the repeats).

Recently, sequence graphs have also emerged into prominence in the growing field of pangenomics, which seeks to analyze the full genomes of many individuals from the same species [4]. In pangenomics, sequence graphs are used to represent genomic variation between individual haplotypes. Sequences in the graph furcate and rejoin around sites of variation so that each individual genome corresponds to a walk through the graph (Fig. 1B). The growth of pangenomics has fueled major advances in both formal algorithms research [21, 12] and practical genomics tools [10, 22].

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

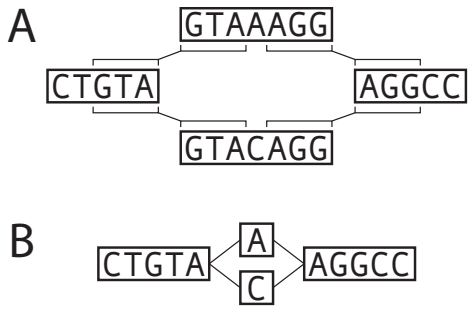

Fig. 1. A: An overlapped sequence graph. **B**: A blunt sequence graph.

Intuitively, the shared basis in sequence graphs should permit the advances in pangenomics to spill over into genome assembly. However, such cross-pollination is stymied by a small difference in the graph formalisms. The edges in assembly graphs indicate sequence overlaps, which are necessary because of the uncertain adjacencies in the underlying genome. In pangenome graphs, the underlying genomes are known, and the edges are *blunt* in that they indicate direct adjacency with no overlap. Blunt sequence graphs can be trivially converted into overlap graphs (with overlaps of length 0), but the reverse requires nontrivial merging operations between the overlapping sequences. As a result, methods have remained siloed within pangenomics despite potential uses in genome assembly.

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

2 Problem statement

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

1. All walks in the overlapped graph are preserved in the blunt graph.
2. Every walk in the blunt graph corresponds to some walk in the overlapped graph.

These two properties prohibit the intuitive solution of transitively merging all overlapped sequences. Doing so can result in walks that are not present in the overlapped graph, because walks can transition between nodes that are not connected by an edge via the transitively merged sequences (Fig. 2). Because overlapped sequences cannot be fully merged, it is necessary to retain multiple copies of some sequences in the blunt graph. However, excessive duplication can create problems for downstream analysis, for instance by increasing alignment uncertainty. Thus, we add one further criterion to the above formulation:

3. Minimize the amount of duplicated sequence.

3 Notation

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

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

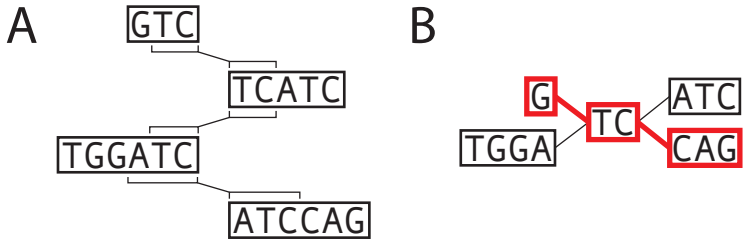

Fig. 2. A: An overlapped sequence graph, and **B**: the blunt sequence graph that results from transitively merging its overlaps. The highlighted walk in the blunt graph does not correspond to any walk in the original overlapped graph.

are a connected by an overlap on s_i's suffix (or vice versa). In the case that a walk traverses a node $s \in S$ from suffix to prefix, we interpret the sequence as its *reverse complement*, which is the sequence of the antiparallel strand of the DNA molecule.

Finally, an *adjacency component* is a collection of affixes (in $S \times \{+, -\}$) that can reach each other via a sequence of adjacent overlaps in O (Fig. 3). This sequence need not form a valid bidirected walk.

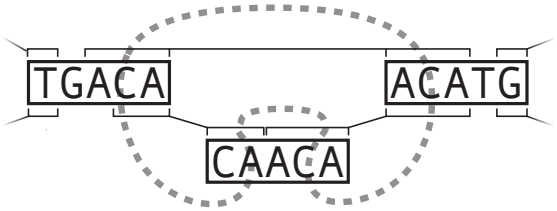

Fig. 3. An adjacency component in a larger sequence graph. Each of the indicated affixes can reach the others by a sequence of overlaps.

4 Methods

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

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

overlaps $(s_1, t_2, +, -)$ and $(t_1, s_2, +, -)$ must also be overlaps in O. Extending this logic, the entire group of overlaps must be contained within a *biclique* subgraph of the adjacency component: two sets of affixes B_1 and B_2 such that every affix in B_1 is connected to every affix in B_2 by an overlap. Thus, we can minimize the number of duplicated sequences by minimizing the number of bicliques needed to cover every overlap edge.

The problem of covering edges with the minimum number of bicliques is known as *biclique cover* (Fig. 4), and it is known to be NP-hard [19]. However, there are domain-specific features of overlapped sequence graphs that often make it tractable to solve large portions of the graph optimally.

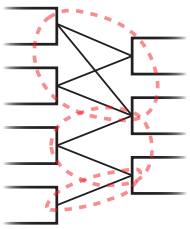

Fig. 4. A biclique cover of an adjacency component with three bicliques.

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

Second, most adjacency components are *domino-free*. This property refers to the absence of a particular induced subgraph, the *domino* (Fig. 5). A sufficient condition to prohibit dominoes is for overlapping to be a transitive property. That is, whenever sequence s_1 overlaps sequences t_1 and t_2, and sequence s_2 overlaps t_1, then s_2 also overlaps t_2. In reality, this is not always the case. However, it is very often the case, since overlaps indicate sequence similarity, and similarity is approximately transitive.

These features guided the design of the following algorithm. If an adjacency component is bipartite and domino-free, we compute the biclique cover in polynomial time with the algorithm of Amilhastre, Vilaren, and Janssen [1]. When an adjacency component is bipartite but not domino-free, we instead use the dual graph reduction algorithm of Ene, et al. [7], followed by their lattice-based post-processing if the algorithm does not identify the optimal solution. Finally, if an adjacency component is not bipartite, we first reduce it to the bipartite case by computing an approximate solution to the maximum bipartite subgraph

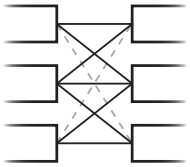

Fig. 5. The domino graph. If either of the dotted edges are present, the induced subgraph is not a domino.

problem using the algorithm of Bylka, Idzik, and Tuza [3]. The maximum bipartite subgraph problem is equivalent to max cut, which is also NP-hard [13]. This process is repeated recursively on the edges that are not included in the bipartite subgraph.

The amount of duplicated sequence is also affected by the manner in which sequences are merged among the overlaps of a biclique. To minimize duplicated sequence, we must maximize matches in the alignment between the overlapped sequences. This is the multiple sequence alignment problem, which is NP-hard. We use the partial order alignment algorithm to approximate the optimal multiple sequence alignment [14]. Partial order alignment also has the advantage that the alignment is expressed as a blunt sequence graph, which can be directly incorporated in the full blunt graph.

5 Implementation

We have implemented the algorithm described here as a genomics tool called GetBlunted. GetBlunted takes as input a GFA file (a common interchange format for sequence graphs [15]) and outputs a GFA containing a blunt graph. In addition, it provides a translation table from sequences in the output to sequences in the input, which can be used to translate analyses performed on the blunt graph into analyses on the overlapped graph. The implementation is written entirely in C++, and it use several auxiliary libraries: GFAKludge is used for manipulating GFA files [5], libbdsg is used to represent sequence graphs [6], and SPOA is used for partial order alignment [24].

6 Results

We compared the performance of GetBlunted to two other tools that transform overlapped sequence graphs into blunt graphs: the gimbricate/seqwish [8, 9] pipeline and Stark [18]. These are, to our knowledge, the only other such tools besides GetBlunted. However, they are not completely comparable. Neither tool provides the guarantees that GetBlunted does for preserving the walk space of the graph. In addition, Stark only works with de Bruijn graphs, a restricted

subset of overlap graphs in which all overlaps are exact matches of a uniform length.

We profiled speed and memory usage on three assembly graphs. The first two are assembly graphs produced by the Shasta assembler [23] for the haploid human cell line CHM13 and for human sample HG002. Both of these were built using Oxford Nanopore reads[1]. The last graph is a de Bruijn graph of Pacific Biosciences HiFi reads of an *Escherichia coli* strain (SRR10382245), which was constructed using jumboDB [2].

All of the bluntifying tools were run on a single core of a c5.9xlarge AWS instance with an Intel Xeon Scalable Processor. Memory usage and compute time were measured with the Unix time tool. The results of the profiling are presented in Table 1. GetBlunted is over 1000 times faster than and comparably memory-intensive to the gimbricate/seqwish pipeline. For de Bruijn graphs, Stark is faster than either tool, although this performance comes at the cost of limited generality.

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

Table 1. Table of speed and memory usage of bluntifing tools run on a single core of an AWS server.

7 Discussion

In this work, we described an algorithm and software tool, GetBlunted, which transforms overlapped sequence graphs into blunt sequence graphs. This provides a route for sequence graph methods developed for pangenomics to be applied to sequence graphs in genome assembly. In both fields, walks through the sequence graph are of primary importance. In genome assembly, some walk through the graph corresponds to the sample genome. In pangenomics, the genomes used to construct the pangenome each correspond to a walk through the graph. GetBlunted provides attractive guarantees that it faithfully preserves the walk space

[1] Publicly available at https://s3-us-west-2.amazonaws.com/miten-hg002/index.html?prefix=guppy_3.6.0/

of the input while also producing parsimonious output. Other comparable methods either do not provide these guarantees or only provide them in limited cases. In addition, GetBlunted is (except in the case of de Bruijn graphs) faster than alternatives that do not provide these guarantees, and it has resource requirements that are easily achievable in any computational environment that is used for genome assembly. In the future, GetBlunted could serve as an step in genome assembly pipelines to improve the quality of their overlap graphs. It could also facilitate direct analyses of assembly graphs in metagenomics applications.

References

1. Amilhastre, J., Vilarem, M.C., Janssen, P.: Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. Discrete Applied Mathematics **86**(2-3), 125–144 (1998)
2. Bankevich, A., Bzikadze, A., Kolmogorov, M., Pevzner, P.A.: Assembling Long Accurate Reads Using de Bruijn Graphs. bioRxiv p. 2020.12.10.420448 (Dec 2020). https://doi.org/10.1101/2020.12.10.420448, https://www.biorxiv.org/content/10.1101/2020.12.10.420448v1, publisher: Cold Spring Harbor Laboratory Section: New Results
3. Bylka, S., Idzik, A., Tuza, Z.: Maximum cuts: Improvements and local algorithmic analogues of the Edwards-Erdos inequality. Discrete Mathematics **194**(1-3), 39–58 (1999)
4. Computational Pan-Genomics Consortium: Computational pan-genomics: status, promises and challenges. Briefings in Bioinformatics **19**(1), 118–135 (2018)
5. Dawson, E.T., Durbin, R.: GFAKluge: A C++ library and command line utilities for the graphical fragment assembly formats. Journal of Open Source Software **4**(33) (2019)
6. Eizenga, J.M., Novak, A.M., Kobayashi, E., Villani, F., Cisar, C., Heumos, S., Hickey, G., Colonna, V., Paten, B., Garrison, E.: Efficient dynamic variation graphs. Bioinformatics (2020)
7. Ene, A., Horne, W., Milosavljevic, N., Rao, P., Schreiber, R., Tarjan, R.E.: Fast exact and heuristic methods for role minimization problems. In: Proceedings of the 13th ACM symposium on Access control models and technologies. pp. 1–10 (2008)
8. Garrison, E.: ekg/gimbricate. https://github.com/ekg/gimbricate (Oct 2020)
9. Garrison, E.: ekg/seqwish. https://github.com/ekg/seqwish (Feb 2021)
10. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., et al.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nature Biotechnology **36**(9), 875–879 (2018)
11. Haubold, B., Wiehe, T.: How repetitive are genomes? BMC bioinformatics **7**(1), 1–10 (2006)
12. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence to graph alignment. bioRxiv (Jan 2019). https://doi.org/10.1101/522912
13. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Springer (1972)
14. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs. Bioinformatics **18**(3), 452–464 (2002)

15. Li, H., Jackman, S., Myers, E., Gonnella, G., Melsted, P., Turner, I., Heuer, M.L., Wilk, J., Minkin, I., Glusman, G., Shcherbin, E., Garrison, E., Dawson, E., Letcher, B., Huang, S., Bolleman, J.: GFA specification. https://https://github.com/GFA-spec/GFA-spec (2013)
16. Myers, E.W.: Toward simplifying and accurately formulating fragment assembly. Journal of Computational Biology **2**(2), 275–290 (1995)
17. Myers, E.W.: The fragment assembly string graph. Bioinformatics **21**(suppl 2), ii79–ii85 (2005)
18. Nikaein, H.: hnikaein/stark. https://github.com/hnikaein/stark (Jan 2021)
19. Orlin, J., et al.: Contentment in graph theory: covering graphs with cliques. In: Indagationes Mathematicae (Proceedings). vol. 80, pp. 406–424. North-Holland (1977)
20. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences **98**(17), 9748–9753 (2001)
21. Rautiainen, M., Marschall, T.: Aligning sequences to general graphs in O(V+ mE) time. bioRxiv p. 216127 (2017)
22. Rautiainen, M., Marschall, T.: GraphAligner: rapid and versatile sequence-to-graph alignment. Genome Biology **21**(1), 1–28 (2020)
23. Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H.E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., Sedlazeck, F.J., Marschall, T., Mayes, S., Costa, V., Zook, J.M., Liu, K.J., Kilburn, D., Sorensen, M., Munson, K.M., Vollger, M.R., Monlong, J., Garrison, E., Eichler, E.E., Salama, S., Haussler, D., Green, R.E., Akeson, M., Phillippy, A., Miga, K.H., Carnevali, P., Jain, M., Paten, B.: Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. Nature Biotechnology **38**(9), 1044–1053 (Sep 2020). https://doi.org/10.1038/s41587-020-0503-6, https://www.nature.com/articles/s41587-020-0503-6, number: 9 Publisher: Nature Publishing Group
24. Vaser, R., Sović, I., Nagarajan, N., Šikić, M.: Fast and accurate de novo genome assembly from long uncorrected reads. Genome Research **27**(5), 737–746 (2017)

Chapter 3

Phased nanopore assembly with Shasta and modular graph phasing with GFAse

Phased nanopore assembly with Shasta and modular graph phasing with GFAse

Ryan Lorig-Roach[1+], Melissa Meredith[1], Jean Monlong[1], Miten Jain[2], Hugh Olsen[1], Brandy McNulty[1], David Porubsky[3], Tessa Montague[4], Julian Lucas[1], Chris Condon[1], Jordan Eizenga[1], Sissel Juul[5], Sean McKenzie[5], Sara E. Simmonds[6], Jimin Park[1], Mobin Asri[1], Sergey Koren[7], Evan Eichler[8], Richard Axel[4], Bruce Martin[6], Paolo Carnevali[6+], Karen Miga[1], Benedict Paten[1+]

[1] UC Santa Cruz Genomics Institute, University of California, Santa Cruz, Santa Cruz, CA, USA
[2] Department of Bioengineering, Department of Physics, Northeastern University, Boston, MA, USA
[3] Department of Genome Sciences, University of Washington School of Medicine, Seattle, WA, USA
[4] The Mortimer B. Zuckerman Mind Brain Behavior Institute, Department of Neuroscience, Columbia University, New York, NY, USA & Howard Hughes Medical Institute, Columbia University, New York, NY, USA
[5] Oxford Nanopore Technologies Inc
[6] Chan Zuckerberg Initiative Foundation, Redwood City, CA, USA
[7] Genome Informatics Section, Computational and Statistical Genomics Branch, National Human Genome & Research Institute, National Institutes of Health, Bethesda, MD USA
[8] Department of Genome Sciences, University of Washington School of Medicine, Seattle, WA, USA & Howard Hughes Medical Institute, University of Washington, Seattle, WA, USA
[+] Corresponding authors: rlorigro@ucsc.edu, pacarnev@ucsc.edu, bpaten@ucsc.edu

Abstract

As a step towards simplifying and reducing the cost of haplotype resolved *de novo* assembly, we describe new methods for accurately phasing nanopore data with the Shasta genome assembler and a modular tool for extending phasing to the chromosome scale called GFAse. We test using new variants of Oxford Nanopore Technologies' (ONT) PromethION sequencing, including those using proximity ligation and show that newer, higher accuracy ONT reads substantially improve assembly quality.

Introduction

Phased genome assemblies enable a variety of clinically-motivated analyses and population studies. For clinical and biological studies, there are many transcriptional and translational outcomes that could result from a given set of mutations in the same gene or regulon, depending on whether they co-occur on the same molecule of DNA[1–3]. Additionally, understanding how variants are linked enables imputation, and therefore a high quality set of phased variants can serve as a catalyst for much larger volume experiments, creating greater

statistical power for disease association[4,5]. Population genetics also benefits from haplotype information because it provides a means to estimate recombination and gene flow[6]. As a consequence of its many applications, a significant portion of recent efforts in human genomics have become focused on generating a high quality, genome-wide set of phased variants[7–9].

Methods for phasing are diverse and can use information from populations or from an individual's sequencing data. At the population level, variants are associated with one another by their co-occurrence across many individuals[9–13]. At the individual level, variants are phased based on co-occurrence in spanning reads. Individual-level methods can be further subdivided into approaches that rely on mapping to an existing assembly and those that use reads directly for creating a phased consensus. When read length or accuracy is limited, mapping-based methods are essential, because mapping is required to find a set of candidate variants that share reads among them. After mapping, reads are usually phased by finding a partition which maximizes the consistency of shared reads among the alleles[14–16].

In contrast to reference-based phasing, *de novo* methods for read-based phasing generate candidate variants internal to the assembler, following read overlap[18,19]. The advantage of assembly-based methods is that they do not fall victim to reference bias, and can therefore identify variants that would otherwise map poorly, as with repetitive regions or large duplications and inversions[20,21]. Reference-based methods can work around this issue by using a draft assembly instead of an existing reference. When parental sequencing data is available, the strategy of trio binning simplifies the problem by partitioning the reads using exact subsequences (k-mers) from parental data. However, its applications have been limited by the added need for parental information and the inability of such informative kmers to adequately label repetitive or homozygous regions which have few haplotype-specific k-mers[17].

Despite the variety of phasing methods and the high demand for phased genomes, all the ONT-specific genome assemblers evaluated at the time of Shasta's original publication (3 years prior to this work), including Shasta, produced collapsed, haploid assemblies. At best, some provided pseudo-haplotypes or alternative contigs which were not explicitly phased or evaluated as such. Since then, none of the remaining ONT-based assemblers have adapted their methods to generate true diploid assemblies. Instead, as a result of the emergence of PacBio HiFi reads[22], modern diploid assemblers have used shorter, more accurate reads to infer haplotype specific overlaps or build a large-k de Bruijn graph[18,19]. In the case of Verkko, ONT reads are used only to resolve regions of the genome graph which remain tangled after construction with HiFi.

To date, there is only one published example of a nanopore assembler which claims to produce phased haplotypes[23], but its repository is not maintained and we were unable to run it to completion. An earlier paper concluded that nanopore *de novo* phasing was not practical at the time[24]. In contrast, the Shasta assembler has kept pace with recent advances in nanopore, and now has a new "Mode 2" of assembly, capable of using R9 or R10 ultra long nanopore reads to phase variation observed in its sequence graph. Following the overlap stage of assembly, Shasta methods and data structures have been replaced in order to explicitly

model sequence “bubbles” or local regions of heterozygous variation and their correlation to one another (see Methods).

Since the basis of assembly phasing lies in the overlapping of reads, highly repetitive or homozygous regions limit phasing in Shasta and other assemblers, such as Hifiasm and Verkko[18,19]. While trio-based haplotagging of partially-phased assemblies produces accurate and global phasing, parental sequencing is not always feasible, and for a variety of species it is essentially impossible. To address this, proximity ligation data has been used for extending phasing beyond the length of a typical read without the need for parental sequencing[25]. Sequencing technologies like Hi-C and Pore-C exploit the physical packing of chromatin in the nucleus to ligate proximal regions of DNA molecules, which can be hundreds of millions of base pairs apart along the chromosome[26,27], exceeding the longest nanopore reads observed[28]. Pore-C is of particular interest because it doesn’t require parental data, it doesn’t need a separate Illumina sequencing machine, and as a result of its protocol it produces many more proximity-based contacts than Hi-C at the same coverage (Figure 1a).

The approach described here leverages the assembly graph to phase and extend partial haplotypes. In this scheme, variants are not inferred from the read alignments. Instead, graph topology or sequence homology is used to identify large scale haplotypic bubbles in the graph, and the information from proximity linkages is used to phase bubbles relative to each other. To find a partition of haplotypes that is consistent with the proximity information, this work uses a new variation of the stochastic optimization methods previously described[18,25]. Once phases are inferred, chaining can then make use of the information stored in the edges of the GFA representation of the assembly graph to achieve a similar result to scaffolding algorithms[29,30]. Our proximity-based phasing methods are evaluated on Shasta, Hifiasm, and Verkko graphs, using both Hi-C and Pore-C data, demonstrating flexibility and reusability. In addition, we show results comparing proximity ligation libraries, which are produced as part of our automated workflow. GFAse also provides a means to do parental k-mer haplotagging using the succinct variation graph that Shasta produces (see Methods).

Using nanopore sequencing for both long reads and proximity ligated reads (Pore-C), we demonstrate a previously undescribed single-sequencer pipeline which is logistically simpler alternative to hybrid and HiFi based approaches, while attaining comparable accuracy (Figure 1b). We test the limits of cost efficiency using a single flowcell of PromethION R10, and attempt to maximize assembly continuity with high coverage ultra-long reads.

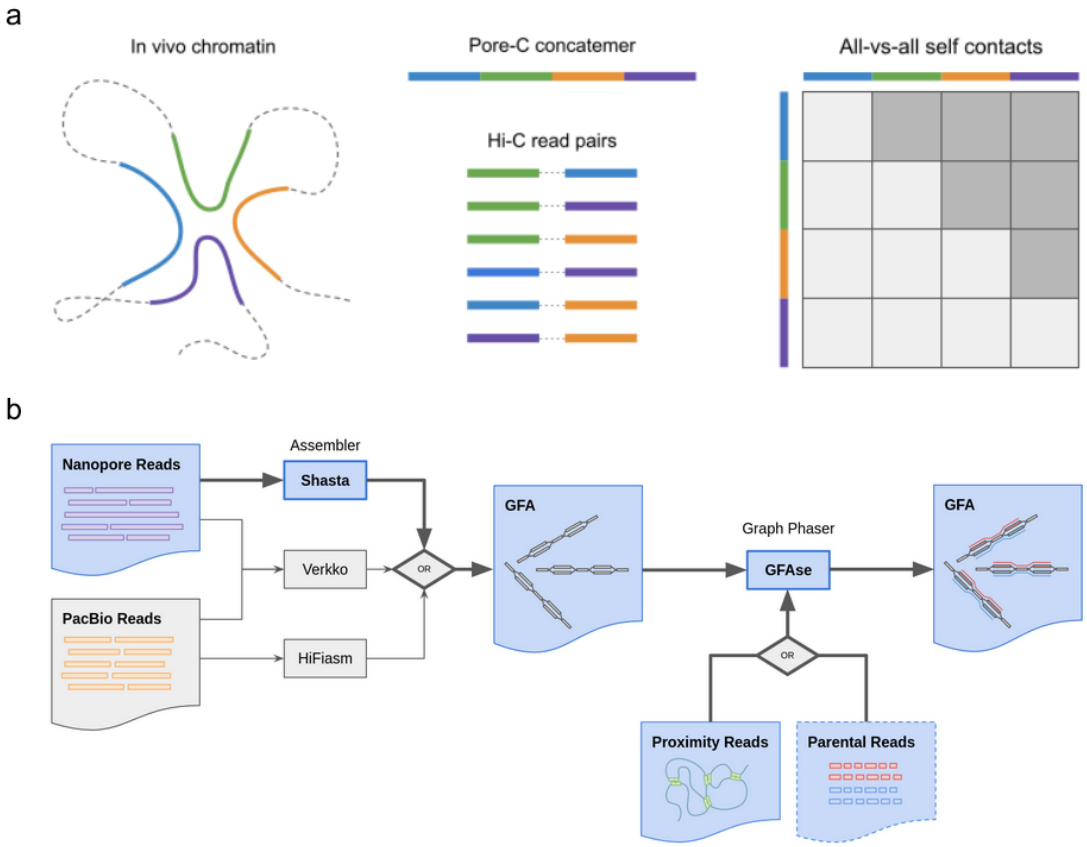

Figure 1: **a)** Diagram of Pore-C sequencing in comparison to Hi-C. In the all-vs-all contact matrix, shaded squares represent usable contacts, which scale at a rate of $1/2n^2$-n for a concatemer of n fragments or subreads. **b)** Summary of *de novo* phasing pipeline using Shasta and GFAse. Shasta performs *de novo* assembly and phases to the extent that is supported by informative variants in the nanopore reads. GFAse then takes a partially phased assembly GFA and extends phasing using orthogonal phasing information. GFAse can perform phasing based on any alignable data type (HiC, Pore-C, etc.). For Shasta graphs, GFAse can also use parental sequencing. The pathways with bolded arrows and blue fill are the methods that are previously undescribed.

Results

Long read sequencing

To address variability in read length, coverage, and accuracy, these results evaluate phasing in 6 different combinations of library preparation and chemistry. The effect of read accuracy on phasing is addressed with differing nanopore chemistries: R9 and R10. For the datasets evaluated, R9 has median accuracy of 95.7-96.1% (Figure 2a) and R10 has 98.3-98.9% median accuracy. In ONT R9, three different length distributions and coverages were assembled. The datasets have minimum read lengths of 10Kbp, 35Kbp, and 100Kbp respectively, so they have been dubbed “standard”, “ultra long” (UL), and “ultra ultra long”

(UUL) for convenience. Nanopore datasets vary considerably in length characteristics, so cumulative length distributions are plotted for proper comparison (Figure 2b).

For R10, a similar series of length distributions are used, with one key difference: the lowest coverage assembly used only a single PromethION flowcell (Figure 2b). For that dataset, reads were sheared to maximize throughput. In the R10 UL dataset, 4 flowcells of unsheared DNA were combined for a high coverage, longer dataset. Finally, in an attempt to maximize contiguity and find the limits of our methods with this data, we have also assembled a dataset containing only reads longer than 165Kbp, from 11 flowcells.

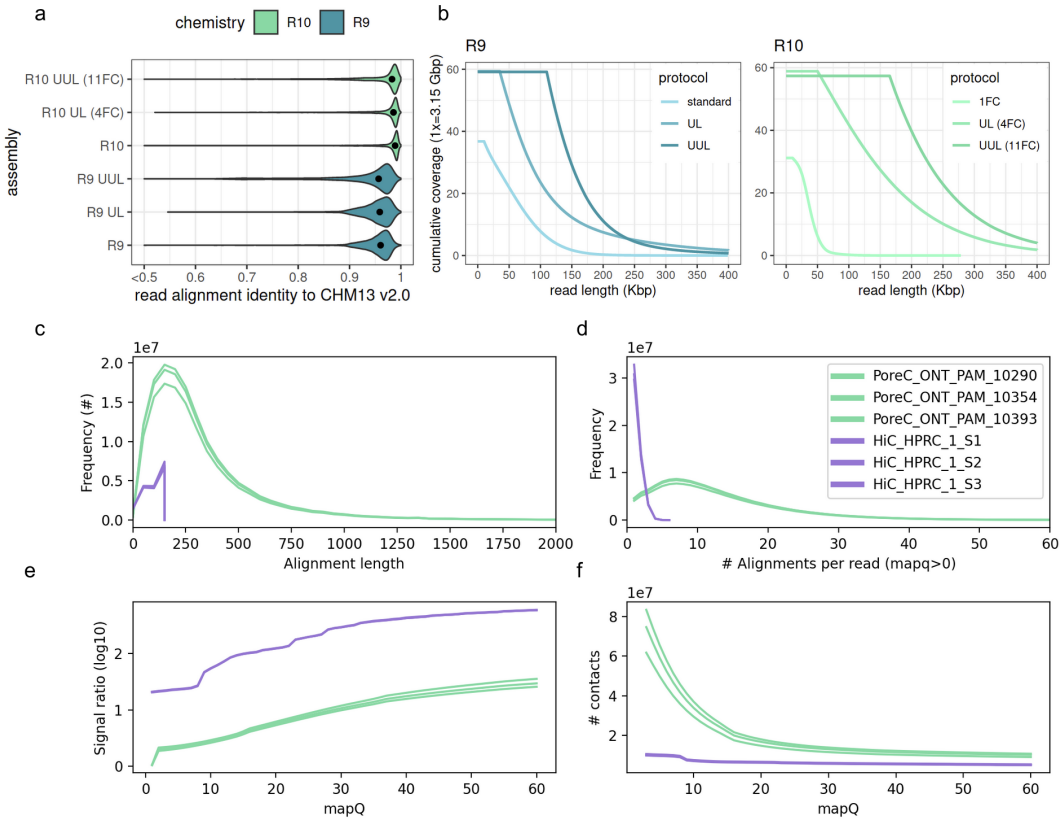

Figure 2: a-b) Identity and length metrics for nanopore read sets used in the HG002 evaluation. **c-f)** Pore-C and Hi-C metrics for contacts and signal ratio, measured on a per-library basis. "Alignment length" and "alignments per read" are proxies for subread statistics. Only mappings which are usable for phasing are shown, i.e with mapping quality (mapQ) >0 in a diploid reference. Signal ratio is computed using a high quality trio-phased assembly to indicate the number of consistent and inconsistent contacts (see methods).

Pore-C and Hi-C sequencing

An early version of the Pore-C protocol was performed and provided by ONT for use in this work, and compared to existing Hi-C datasets from the HPRC Year 1 data freeze. Notably, these Pore-C libraries do not have a larger modal alignment length than Hi-C (Figure 2c). The difference between Pore-C's multi-contact concatemers and Hi-C's 1-to-1 paired

ligation are shown using alignments. Pore-C concatemers are composed of many smaller reads, or “subreads”, analogous to Hi-Cs paired reads. As a result of having many subreads, each Pore-C read has many alignments, and its contacts accumulate in an all-vs-all manner among subreads. The number of subreads (usable for phasing) can be in the dozens (Figure 2d). In contrast, Hi-C accumulates at most one long range contact per pair of reads, and in many cases, read pairs contain unmappable reads, which drastically reduces its throughput.

To demonstrate the practical differences between Pore-C and Hi-C that are particularly relevant to phasing, these results also show the signal ratio (Figure 2e) and the total number of contacts (Figure 2f) for reads which map to a diploid reference. Since mapping quality is used to filter contacts during phasing, the spectrum of signal ratio and number of contacts is plotted across observed map qualities. Signal ratio is computed using a trio-phased diploid reference, to estimate the number of consistent and inconsistent contacts (cis or trans w.r.t. haplotypes). In summary, contacts from Hi-C consistently have a higher signal ratio, while Pore-C produces more contacts.

Assembers evaluated

For phasing and assembly quality evaluation, we compare to widely-used pipelines for diploid assembly that use ONT reads, PacBio CCS, or both. For PacBio CCS and hybrid CCS/ONT assemblers we have included Hifiasm and Verkko (Table 1). As the name implies, Hifiasm is an assembler for PacBio CCS (HiFi) data which has built-in methods for trio and Hi-C phasing. When comparing Shasta to Hifiasm the relative strengths of PacBio and nanopore become apparent. GFAse is also compared to Hifiasm’s native phasing methods to evaluate GFAse’s performance. Verkko is used in this comparison as an upper limit, since it uses both high coverage CCS and ONT reads to generate its assemblies. In this comparison, Hifiasm uses 30x coverage CCS and 30x coverage HiC. Verkko “production” assemblies use 42x CCS and 70x ONT >100Kbp, and trio Illumina. The “full coverage” Verkko assembly uses 190x CCS.

For nanopore, HapDup has recently developed a combination of alignment-based and assembly-based methods for phasing long reads[31]. HapDup uses a linear unphased Shasta assembly as a starting point, phases aligned reads, and then generates a reference-free consensus for each haplotype. HapDup is specialized for structural variant detection in low coverage nanopore datasets, so it is a natural comparison point for phased Shasta assemblies.

		ONT		**CCS**		**HiC / PoreC**
Assembler	**Label**	**Coverage**	**N50 (Kbp)**	**Coverage**	**N50 (Kbp)**	**Coverage**
HapDup (Shasta)	"	38	31			
Shasta	R9 Standard	37	60			45/30
Shasta	R9 UL	60	80			45/30

Shasta	R9 UUL	60	150			45/30
Shasta	R10 (1FC)	26	32			45/30
Shasta	R10 UL (4FC)	60	130			45/30
Shasta	R10 UUL (11FC)	~60	230			45/30
Hifiasm Trio	"			30	17.5	
Hifiasm Hi-C	"			30	17.5	30
Verkko Trio	production	185.77	81.20	42.66	14.75	
Verkko Trio	full coverage	971.71	50.65	169.03	17.22	

Table 1: Coverage summaries for HG002 assemblies evaluated in this analysis

Phasing results

To evaluate phasing accuracy, the assemblies presented are aligned to a common reference and their heterozygous alleles are compared using WhatsHap to an orthogonally phased truth set, produced by NIST's Genome In A Bottle[32] consortium. Switch rate indicates how often alleles in the sample switch phase relative to the truth set, and hamming rate indicates the proportion of switched loci. Genotypes with allele sequences that do not both exactly match the reference VCF are not evaluated for phasing, which is accounted for by reporting the number of variants covered.

GFAse Trio uses k-mers from parental Illumina short reads to phase the heterozygous bubbles in the child Shasta assembly graph. Phasing results show that Shasta + GFAse Trio using Illumina reads outperforms Hifiasm Trio and Hifiasm HiC in terms of median switch rate (~ 0.0005) and hamming error (~ 0.0005). Shasta + GFAse trio results are also within range of the Verkko Trio 'production' assembly that uses both ONT and PacBio input reads. Two chromosomes, Chr15 and Chr16, have higher hamming rates across all the R10 Shasta assemblies, which may be due to the difficulty of resolving these chromosomes and identifying inversions relative to the hg38 reference. This method requires short read sequencing of both parents which, for various reasons, may not always be feasible or cost effective. For the more contiguous Shasta assemblies, trio results closely match the phasing results from Hi-C and PoreC phasing which don't require any parental sequencing.

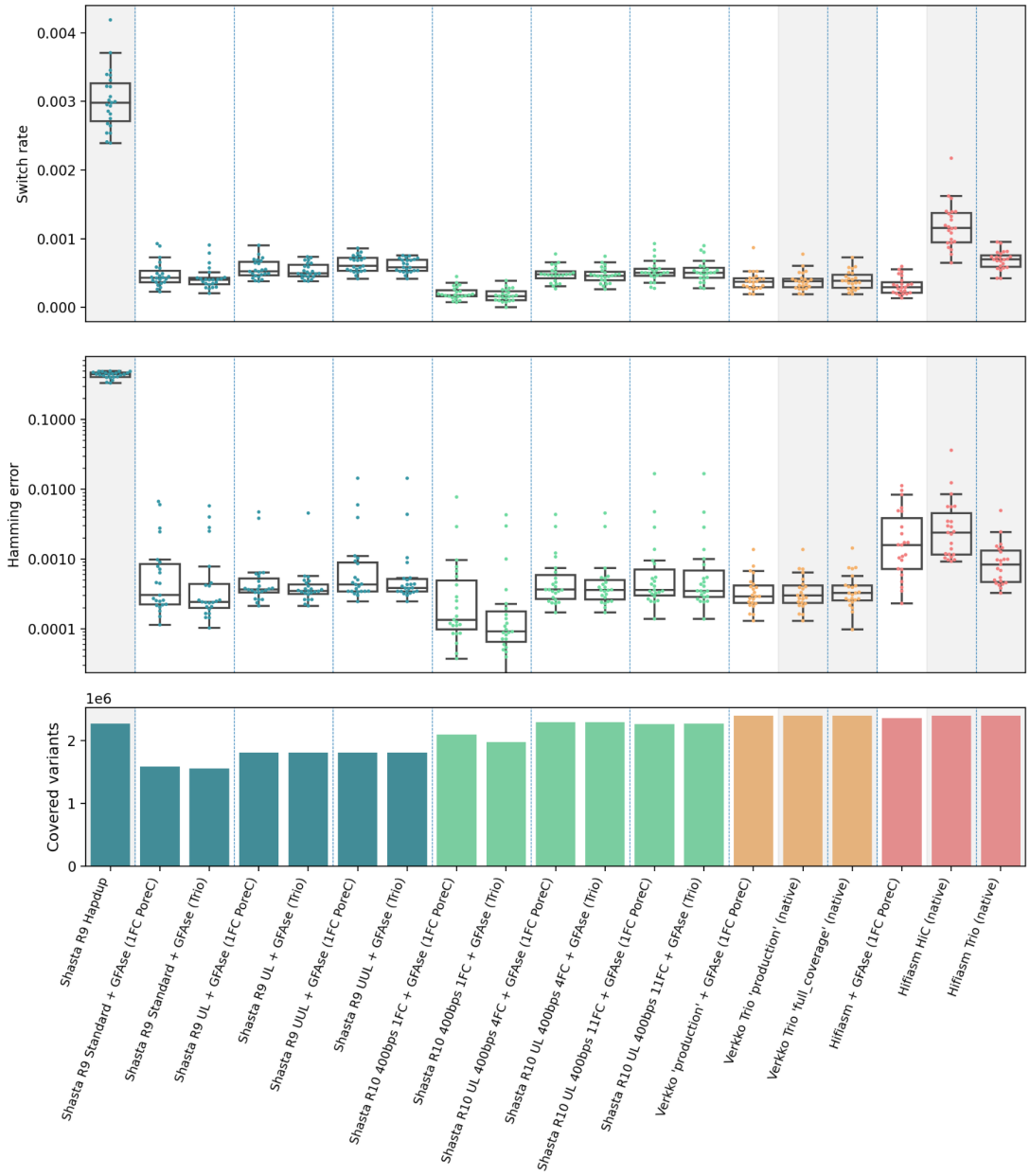

Figure 3: Phasing metrics for HG002 assemblies, as evaluated using the GIAB v4.2.1 benchmark VCF, phased with StrandSeq using WhatsHap (see methods). All shasta assemblies are unpolished. Assemblies not phased with GFAse are shaded gray. Each dot represents a chromosome error rate, generated by WhatsHap compare. Native Hifiasm HiC uses 30x coverage. Each pair of HiC is ~17x. PoreC flowcells have ~30x yield.

Shasta + GFAse assemblies consistently outperform native Hifiasm HiC phasing, and in some cases, hamming rates match or beat trio-phased Hifiasm. When comparing the total number of assessed variants, R10 assemblies drastically improve on R9, likely as a result of its lower error rate. This effect can be seen by comparing the standard length results to UL, in both R9 and R10. It is also evident from the HapDup results that polishing has a drastic effect on covered variants, since it uses a single flowcell R9 Shasta assembly. UL R10 assemblies reach nearly as many variants covered as Verkko while also producing comparable switch and hamming rates.

Yak Trio eval, an orthogonal evaluation method that uses parental short read k-mers to evaluate phasing accuracy, reports an order of magnitude higher switch and hamming error for Shasta assemblies compared to Verkko (Supplementary Table 2). This could be from a combination of greater genome coverage and systematic false positives in the switch analysis. As one possible explanation for the discrepancy between yak and WhatsHap, we observed that some of the yak switch blocks occur in regions of the Shasta assembly that differ from Verkko only in homozygous variants (Supplementary Figure 4a) and this is particularly evident in chrX of the male HG002 assembly (Supplementary Figure 4d). As an additional source of enrichment of errors, we observe that some yak switches occur in repetitive regions that contain homozygous homopolymer errors (Supplementary Figure 4c). When comparing Shasta Dipcall VCFs directly to Verkko Dipcall VCFs we observe a variant coverage of 2.48M at a hamming rate of 0.0003, but the total variants from Dipcall differ significantly in assembly specific Indels (Supplementary Table 3).

As evaluated by WhatsHap, GFAse consistently produces a median chromosomal hamming error of <0.001% for Shasta assemblies, which is reduced from an average chromosomal hamming of 40-50% for non-globally phased assemblies. In the single flowcell, standard length R10 experiment, R10 produces shorter haplotype bubbles than R9, which are then phased together at the chromosome scale with proximity ligation data (Figures 3 and 4d). In the high coverage UL R10 assembly, a large portion of the variants exist in continuous haplotypes directly from the Shasta assembler (Figure 4e). Generally, the graphs with high input N50 reach trio-level phasing accuracy when phased with HiC. The highly fragmented input graphs such as the Shasta “standard” R9 or the Hifiasm PacBio graphs converge to a less optimal phasing.

In addition to this analysis, a limited number of HG005 R9 and R10 assemblies were evaluated (see Supplementary Figure 2 and Table 1), along with 4 HPRC R9 assemblies (Supplementary Figure 3). A similar trend in phasing accuracy is observed across all additional individuals, with the caveat that there is no StrandSeq-derived truth set for the HPRC assemblies.

By comparing Pore-C and Hi-C (Supplementary Figure 1) it is evident that more information is provided by one library of Pore-C than Hi-C. It takes 3 pairs (6x2 lanes, or ~45x) of Hi-C to reach a phasing that is roughly equivalent to 1FC Pore-C (30x). This is likely due to the higher number of total contacts per Pore-C read. In combination with the single-flowcell R10 assembly, we achieve an accurately phased human assembly with a total of 2 PromethION flowcells.

Assembly quality

Structural variants were evaluated with Truvari and the GIAB HG002 Tier1 structural variant collection as a truth set. Shasta nanopore assemblies consistently yield structural variant F1 scores above 90%, with a peak score of >95% in the UL 4FC and 11FC R10 assemblies. Most notably, a single flowcell of R10 400bps nanopore data can reach an F1 score of around 94% (Figure 4a). HapDup[33], which starts with Shasta assembly, uses a local realignment and polishing to recover short collapses, which is a probable cause for its greater recall. Precisions from R10 assemblies match or exceed HiFi-based methods, but recall values do not, most likely as a result of collapse in the repetitive regions of the genome.

A similar trend is seen in the gene level analysis performed by asmgene (Figure 4b), in which the number of full single copy genes are comparable to HiFi or hybrid assemblies, but multi copy genes are reduced in comparison. For gene completeness, the highest scoring Shasta assembly reached ~75% multicopy genes at a 99% identity threshold, which is now 10% lower than Hifiasm. In terms of base level quality, R9 assemblies have base qualities greater than Q30, while R10 assemblies are greater than Q40 and PacBio HiFi or hybrid assemblies exceed Q50 (Figure 4c).

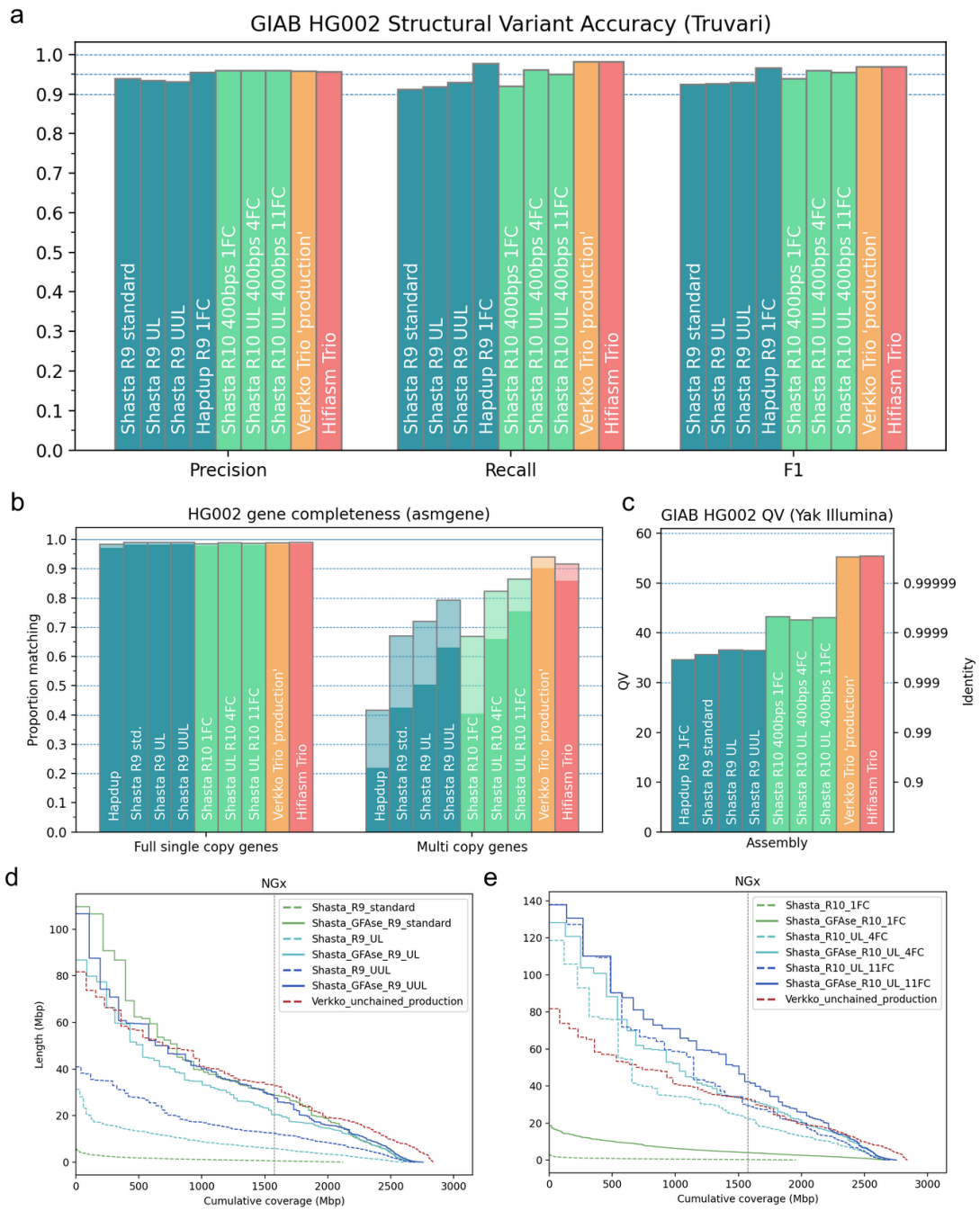

Figure 4: Structural variant, base level, and gene level accuracy metrics for HG002 assemblies. **a)** Base accuracy evaluated using yak with Illumina NovaSeq. **b)** Gene completeness measured by asmgene using human transcript sequences. "Full single copy" genes only indicate unfragmented, non-duplicated genes, matching transcripts by 99% or greater coverage and stratified by >97% (translucent) or >99% identity (opaque). Multicopy genes are similarly stratified. **c)** SVs evaluated using the GIAB Tier1 benchmark VCF with Truvari. **d-e)** NGx Plots for Shasta haplotypes, before and

after unzipping bubble chains with GFAse. For comparison, the phased portion of the un-chained Verkko 'production' assembly is shown. The vertical line indicates the NG50 for each assembly.

Shasta assemblies are chained and unzipped by GFAse to achieve greater continuity. In theory, only 0.1% of the assembly would need to be assembled as diploid bubbles to produce a fully phased assembly after unzipping, but in practice, mapping and optimization with proximity ligation reads is easier when haplotypes are long. Before chaining, bubble N50s range from <1Mbp in the low coverage experiments up to 39.7Mbp in the highest quality assembly. For the various R9 assemblies, the chained and unzipped assemblies remain consistent in length despite their variable input lengths, which is shown by their largely overlapping post-GFAse NGx distributions (Figure 4d-4e). R10 sequencing protocols are currently limited by throughput, so sheared reads were used for the single flowcell experiment, which do not enable the same level of contiguity as the UL assemblies. However, for the higher cost 4FC and 11FC UL experiments, the upper limit on contiguity exceeds our previous most contiguous R9 assemblies.

Non-human assemblies

To test Shasta's phased assembly methods outside the context of human genomes and basecalling, two species were assembled: the dwarf cuttlefish (*Sepia bandensis*) and the broad bordered yellow underwing moth (*Noctua fimbriata*). *S. bandensis*, previously unsequenced by long reads, was sequenced for this work to a depth of ~105x with 30x >100Kbp length. *N. fimbriata* was previously sequenced by the Darwin Tree of Life[34] to a depth of 87x with an N50 of 28.7Kbp.

In non-human assemblies, truth sets are limited, so this analysis relies on BUSCO to evaluate gene completeness and extent of phasing (Table 2). By comparing the BUSCO score for the full diploid assembly, and one haplotype of the diploid assembly, the number of phased and unphased genes is measurable. In both species presented, complete phased genes are estimated to range from 86% to 89% of the 954 genes in the metazoan dataset.

	Complete and single-copy (S)		**Complete and duplicated (D)**				
	Both haps	**One hap**	**Both haps**	**One hap**	**Total diploid**	**Total Haploid**	**Diploid N50**
Sepia bandensis	5.97%	95.39%	90.15%	0.73%	5330371146	415993994	3794550
Noctua fimbriata	7.02%	93.29%	91.40%	4.82%	488722705	21756252	5094850

Table 2: Non-human Shasta phasing metrics, in terms of BUSCO gene completeness. These results use the metazoan dataset, which has 954 genes.

Resource usage

For the slowest assembly evaluated (4 flowcell R10 UL), Shasta runs in 12 hours on a 64 thread 1.2TB AWS instance. This allows for 14 assemblies to be run in the same amount of core hours as a single Verkko assembly (Table 3). For unfragmented assemblies, GFAse has variable run time of 2.3hr to 4.6hr using 64 threads, which depends on the number of contacts in the alignments. In the worst case scenario, with a highly fragmented GFA such as the unphased Hifiasm graph, and high contact dataset such as PoreC, GFAse can take up to 12 hours.

	CPU hours	Est. wall hours (64 vCPU)	Peak RAM (GB)
Verkko	8288.57	129.5	61
Hifiasm	366.9	5.7	150
HapDup	2425.0	24.0	624
Shasta (UUL)	582.4	9.1	1283
Read alignment	307.2	4.8	140
GFAse (HiC)	75.0	2.3	84
GFAse (PoreC)	145.7	4.6	84
GFAse (Trio)	0.58	0.58	30

Table 3: Run time performance for various assemblers presented in this paper. Separate times shown for GFAse HiC and PoreC as a result of its dependence on the number of contacts. For HapDup, CPU hours and wall hours show the sum of multiple steps in the HapDup pipeline, including an initial Shasta assembly, which used 96 cores. GFAse trio runs single threaded.

Discussion

In this work, we use the latest advances in nanopore sequencing to simplify the challenge of producing phased *de novo* assemblies. We demonstrate accurate phasing in a two PromethION flowcell pipeline, using one long read flowcell and one Pore-C flowcell. Phased contiguities yielded by our higher coverage experiments are unmatched in previous nanopore assemblies, and Pore-C as a phasing data type is unexplored in prior publications. The tools presented are efficient and modular, and they rely on sequencing protocols which have a relatively short turn around[35], enabling rapid prototyping.

We focus on nanopore sequencing because, despite its currently lower base-level accuracy relative to competitors, its ability to sequence native DNA means that its upper read length is essentially limited only by the library preparation and loading procedure, which gives it a unique advantage over methods which rely on synthesis or amplification[36]. With recent changes in chemistry and computational methods, this tradeoff in accuracy has

reduced, while cost and throughput have improved. In three years, since Shafin et al 2020, R9 median accuracy has increased from 90% to 95%, effectively reducing error by half. In the same time span, protocols for nanopore library preparation have improved average N50s from 42Kbp to >100Kbp, more than doubling. Now, with the R10.4.1 chemistry in production, we see yet another reduction in error, bringing accuracy into the 98-99% range.

The alternative sequence type to nanopore, PacBio HiFi (CCS), has accuracies of >99.9%[22], but its reads are size-selected, ranging from 10Kbp to 30Kbp. This means that they have the accuracy to distinguish copies of less diverged repetitive units in the genome, but not necessarily the length to span them. Hybrid assemblers have integrated both ONT and HiFi data to accommodate for this, in addition to parental ("trio") Illumina data or Illumina proximity ligation data (Hi-C) to assist with phasing. Hybrid methods have achieved unprecedented accuracy and contiguity by leveraging the strengths of 3-4 different sequencers, but as a result they are also costly in terms of resources, logistics, and time.

Using technological advances in ONT sequencing, our results produce notable improvement over the previous standards for nanopore phasing. Evaluation metrics for phased assembly are approaching that of the hybrid assemblers in gene completeness, contiguity, and phasing accuracy, showing promise for future single sequencer pipelines. When considering that all Shasta assemblies generated for this paper are unpolished, it is clear that there is room for further improvement. Contiguity and completeness are likely to continue to grow proportionally with additional throughput of long reads. To make full use of the longest reads, further method development is underway to address repetitive regions which defy the diploid assumption of our current methods. The consistent trajectory of ONT quality and throughput has motivated our work, and we aim to continue to adapt long read assembly methods to future improvements.

From a software development perspective, the tools presented in this paper are written with the goal of modularity, interpretability, and flexibility of usage. The fully specified graph outputs of Shasta make it an ideal resource for downstream development and analysis, as is demonstrated by the application of GFAse in this context. GFAse employs transparent and reusable data structures, and similar to Shasta, produces comprehensive outputs that describe the homology, proximity linkage, and inferred haplotype chains in the graph. Relying directly on alignments, GFAse is capable of using any data type for phasing which can be aligned to the assembly in BAM format. In theory, long reads, or even conventional linked reads could be used as phasing information if their alignments span the unphased regions of an assembly. This makes it a flexible module for future applications, as long range linked data types continue to evolve.

Methods

Shasta - *De novo* phasing with nanopore reads

Shasta is an assembler specialized for nanopore reads, and it uses the Overlap Layout Consensus paradigm of assembly. It starts by reducing reads into vectors of fixed length subsequences or markers, and then it computes an approximate overlap among reads using a variation of the MinHash algorithm[37]. Shasta refines its candidate overlaps using alignment in marker space, and reduces the overlap graph by filtering alignments and creating a k-nearest neighbor graph. For more details, see the initial 2020 publication, or the online documentation[35,38].

In this updated version, Shasta performs *de novo* phasing internally, using only conventional nanopore reads. Shasta uses a data structure referred to as a "phasing graph", built from a marker representation of the reads. The phasing graph is created following the overlap stage of assembly, and it describes the coverage in terms of read IDs covering each branch of a heterozygous diploid bubble found in a graph. For each pair of bubbles, a Bayesian model computes the probability that they are either uncorrelated or in one of two possible phased orientations with respect to each other. By iteratively aggregating bubbles with this Bayesian criteria for correlation, groups of phased bubbles are established, while uncorrelated error bubbles remain isolated.

Given a set of phased bubbles or a "component", Shasta then identifies local bubble chains within each set, which are bubbles in series, constituting collinearly traversable regions of the graph[38]. These bubble chains have a strict topology in which elements in the chain can either be homozygous unphased nodes or heterozygous phased pairs of nodes (see Figure 4A). Bubble chains are the basis for subsequent unzipping into haplotypes. For completeness, Shasta generates output GFAs containing the succinct representation of edits, or "Detailed" graph, as well as the larger unzipped haplotype representation, or "Phased" graph. Both of these representations contain bubble chains with differing length haplotype sequences. The "Phased" representation is convenient for downstream phasing because its sequences tend to greatly exceed the length of a read and multiple variants can be spanned with conventional mapping. On the other hand, the "Detailed" representation summarizes the edits between phased haplotypes, and represents longer scale phasing using a path in the GFA. Graphs generated by Shasta contain "blunt" or non-overlapping nodes, which makes chaining them trivial.

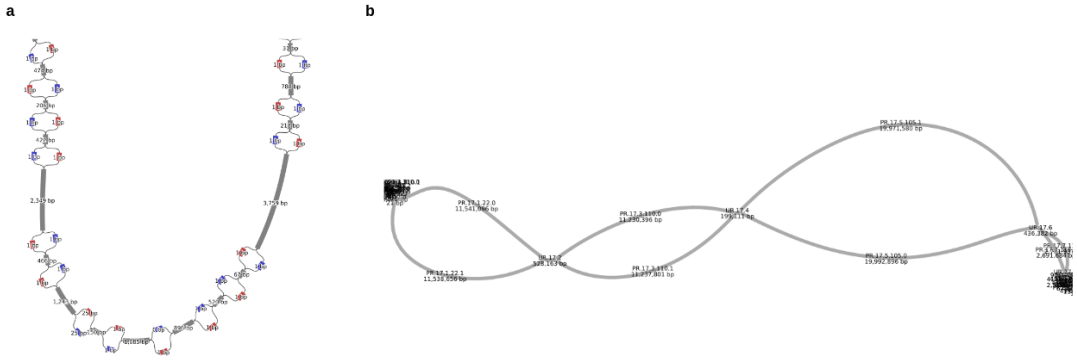

Figure 4: The two types of Shasta output graphs, visualized as a 2D layout in Bandage[39] at two different scales. **a)** A subregion of the "Assembly-Detailed.gfa", showing a near-variant scale nodes in a bubble chain and their phasing indicated by colors produced by Shasta. **b)** A subregion of the "Assembly-Phased.gfa" showing a phased portion of chr11 from HG002 which terminates at two tangles, presumably caused by telomeric and centromeric sequences.

GFAse - Phasing graphs with proximity ligation data

GFAse relies on conventional mappings for phasing information. HiC, PoreC, or other proximity-ligated reads are mapped to the GFA contigs using whichever mapper is most appropriate for the sequence type. The strength of the proximity linkage between any two contigs is updated as a sum, in the form of a weighted edge in a "contact graph", where the weight is the number of reads linking them. Uninformative mappings which do not cover a heterozygous site are filtered by setting a map quality threshold.

To phase the graph, GFAse first identifies haplotypic bubbles. Two methods are available in GFAse: assembler annotation and sequence similarity search. Efficient similarity search is accomplished with a variation of MinHash[37] similar to that used by Shasta[35], and then refined with full scale alignment with minimap2[40].

Phases are optimized using a stochastic method which approximates a solution to the optimization variant of the max-cut problem[15,18,25]. The method depends on an objective function which penalizes inconsistent contacts and rewards consistent contacts. If any two bubbles are compared, there are four possible contacts, and only contacts linking the contigs in matching phases have positive scores. For GFAse, a variation on existing methods was introduced to improve reproducibility of the stochastic method and perform better on fragmented graphs in which the state space is much larger. In short, the method takes samples from repeated greedy optimizations of randomly initialized phase states and accumulates a distribution of orientations, which is then used to merge bubbles that are most consistent. One benefit of sampling many times with few iterations is that samples are independent and can be multithreaded.

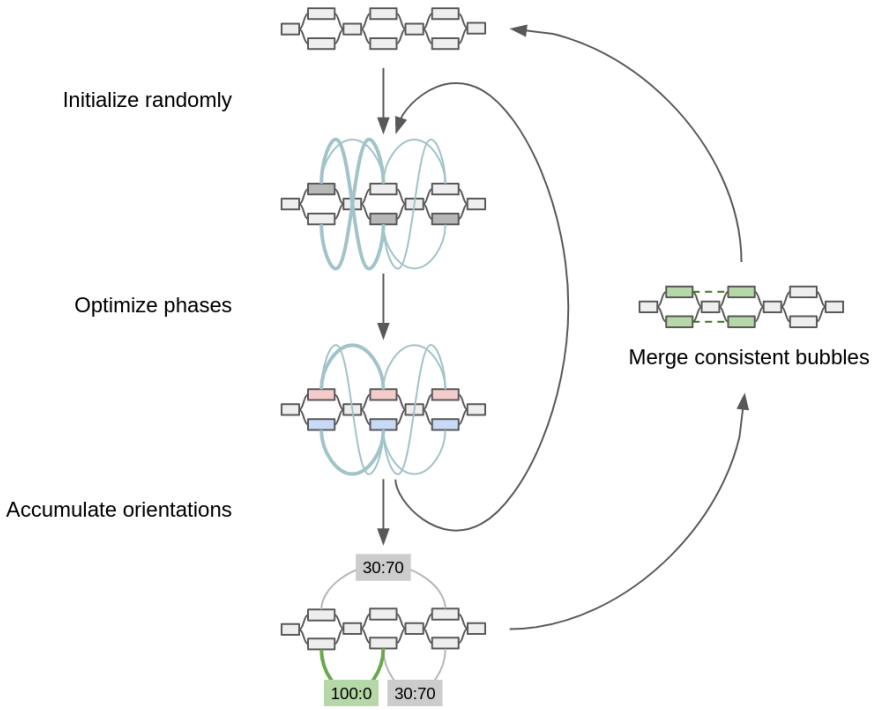

Figure 5: Diagram of sampling method for optimizing proximity linkages in an assembly graph. Edge weights in the contact graph are represented by teal curves. For each inner iteration, a greedily converged phase state is used to update a distribution of orientations among bubbles. Bubbles with the strongest signal at the end of sampling are merged for successive iterations. By the end of each round *r* of merging, the largest possible bubble set is 2^*r* in size.

GFAse - Phasing with parental data

Homozygous parental k-mers are selected from each parent and used to phase the "detailed" assembly GFA by counting parental k-mers in the heterozygous bubbles. To process the parental sequence data, reads are broken into 31 base pair k-mers using kmc3[41]. Kmc3 subtract was used to identify k-mers that are unique to each parent. Finally, unique homozygous k-mers are matched to child k-mers on heterozygous bubbles assigning a phase to bubble components, using a simple majority vote. Illumina reads for the HG002 Ashkenazi Jewish Trio sample were obtained from the publically available 1000 Genomes Project (fc-4310e737-a388-4a10-8c9e-babe06aaf0cf/working/HPRC_PLUS/HG002/raw_data/Illumina/parents/HG003 and fc-4310e737-a388-4a10-8c9e-babe06aaf0cf/working/HPRC_PLUS/HG002/raw_data/Illumina/parents/HG004).

GFAse - Chaining phased graphs

With a phased assembly graph, adjacent bubbles are chained in a manner similar to scaffolding, to extend haplotypes. GFAse first loads the GFA using the VG HandleGraph data structure[42] and identifies tractable regions as anything which follows a strict diploid bubble chain topology. Diploid nodes have exactly one two-hop neighbor and, at most, two direct adjacencies in each direction. Chains are then identified by traversing contiguous subgraphs of labeled nodes. With bubble chains identified, haplotypes are labeled with paths in the GFA formalism. Then they can be "unzipped" trivially by traversing them and duplicating the homozygous nodes into both haplotypes. The strict definition of bubble chains used in this method is intended to maximize fidelity to the input graph by reducing misjoins in the chaining step.

Sequencing and data acquisition

R9 Standard
Sequencing was performed as described in Shafin et al. 2020, and re-basecalled with Guppy v5.0.7.

R9 UL
ONT data from the GIAB consortium[43] was re-basecalled with Guppy >v5.0 and combined with the R9 Standard dataset to provide longer reads.

R9 UUL
DNA extractions from 6 million cells of HG002 were prepared using Circulomics Nanobind CBB Kit (Pacific Biosciences, 102-301-900). Libraries were prepared using Ultra-Long DNA Sequencing Kit (SQKULK001). The libraries were sequenced on Flowcell R9.4.1 on PromethION for 72 hours. Flowcells were washed using the Flowcell Wash Kit (EXP-WSH004) every 24 hours. Fresh libraries were loaded after each wash.

R10
DNA extractions from 5 million cells of HG002 were prepared using PacBio Nanobind CBB kit (SKU 102-301-900) according to UHMW DNA Extraction Cultured Cells protocol (EXT-CLU-001). Standard pipette tips were used to generate a homogenous sample before DNA shearing. DNA was sheared to a target size of 50kb on Megaruptor® 3. Samples were normalized to 100 µL volume at 50 ng/µL concentration and sheared at speed 27 using Megaruptor shearing kit (E07010003). DNA size was assessed post shearing on Agilent Femto Pulse System using gDNA 165kb Analysis Kit (FP-1002-0275). Post shearing, DNA size selection was performed using PacBio SRE kit (SKU 102-208-300) following manufacturer's recommendations. Libraries were prepared using Oxford Nanopore Ligation Sequencing Kit V14 (SQK-LSK114) according to Oxford Nanopore protocol

GDE_9161_v114_revK_29Jun2022. The libraries were sequenced on Flowcell R10.4.1 on PromethION for 96 hours. Flowcells were washed using the Flowcell Wash Kit (EXP-WSH004) every 24 hours. Fresh libraries were loaded after each wash.

R10 UL
Nanopore sequencing dataset were generated following Oxford Nanopore protocol ULK_9177_v114_revC_27Nov2022. DNA extractions from 6 million cells of HG002 were prepared using Monarch® HMW DNA Extraction Kit for Tissue (New England Biolabs, T3060). Libraries were prepared using Ultra-Long DNA Sequencing Kit (SQKULK114). The libraries were sequenced on Flowcell R10.4.1 on PromethION for 72 hours. Flowcells were washed using the Flowcell Wash Kit (EXP-WSH004) every 24 hours. Fresh libraries were loaded after each wash.

Sepia bandensis
Testes tissue (3-5 mg) from an adult male dwarf cuttlefish was homogenized in PBS using a Dounce homogenizer. This was followed by DNA extraction using Circulomics Nanobind Tissue Kit (Pacific Biosciences, 102-302-100). Libraries were prepared using Ultra-Long DNA Sequencing Kit (SQKULK001). The libraries were sequenced on Flowcell R9.4.1 on PromethION for 72 hours. Flowcells were washed using the Flowcell Wash Kit (EXP-WSH004) every 24 hours. Fresh libraries were loaded after each wash.

Noctua fimbriata
ONT data was acquired from the Darwin Tree of Life project[34].

Generating assemblies

To generate nanopore assemblies, Shasta (v0.10.0 unless otherwise specified) was run with the appropriate configuration for each data type, as follows:

R9 Standard
--config Nanopore-Phased-May2022
--Reads.minReadLength 10000
--Assembly.mode2.phasing.minLogP 30

R9 UL
--config Nanopore-UL-Phased-May2022
--Reads.minReadLength 10000
--Reads.desiredCoverage 180000000000
--Assembly.mode2.phasing.minLogP 50

R9 UUL
--config Nanopore-UL-Phased-May2022
--Reads.minReadLength 110000
--Assembly.mode2.phasing.minLogP 50

R10 (1FC)
--config Nanopore-Phased-R10-Fast-Nov2022
--Assembly.mode2.phasing.minLogP 20

R10 UL (4FC)
Configs for *Nanopore-Phased-R10-Fast-Nov2022* and *Nanopore-UL-Phased-May2022* were merged, with *Nanopore-Phased-R10-Fast-Nov2022* taking precedence for any conflicting parameters. The following parameters were then added:
--Assembly.mode2.phasing.minLogP 20
--Reads.minReadLength 50000

R10 UUL (11FC)
(Shasta v0.11.1)
--config Nanopore-Phased-R10-Fast-Nov2022
--Kmers.probability 0.05
--MinHash.minBucketSize 20
--MinHash.maxBucketSize 60
--Align.minAlignedMarkerCount 2500
--Reads.minReadLength 170000

Sepia bandensis
--config Nanopore-UL-Phased-May2022
--Reads.desiredCoverage 400G

Noctua fimbriata
--config Nanopore-Phased-May2022

Phasing with GFAse

To phase with GFAse (https://github.com/rlorigro/GFAse), reads are first aligned to contigs using conventional mapping and alignment. Paired HiC reads were aligned using BWA-mem (https://github.com/lh3/bwa) and PoreC concatemers were aligned using minimap2 (https://github.com/lh3/minimap2). A WDL which combines these steps can be found at https://github.com/meredith705/gfase_wdl/tree/main .

PoreC data was aligned using minimap2 with the following parameters:

```
minimap2 \
-a \
-x map-ont \
-k 17 \
-t 56 \
-K 10g \
-I 8g \
Assembly-Phased.fasta \
porec_reads.fastq.gz \
| samtools view -bh -@ 8 -q 1 - \
> porec_to_assembly.sorted_by_read.bam
```

HiC data was aligned using BWA with the following parameters:

```
bwa index Assembly-Phased.fasta \
&& \
bwa mem -t 46 -5 -S -P \
Assembly-Phased.fasta \
HG002.HiC_1_S1_R1_001.fastq \
HG002.HiC_1_S1_R2_001.fastq \
| samtools sort -n -@ 24 - -o hic_to_assembly.sorted_by_read.bam \
```

Shasta assemblies were phased with HiC using the following parameters:

```
/home/ubuntu/software/GFAse/build/phase_contacts_with_monte_carlo \
-i hic_to_assembly.sorted_by_read.bam \
-g Assembly-Phased.gfa \
-o /path/to/output/directory/ \
-m 1 \
-t 62
```

Shasta assemblies were phased with PoreC using the following parameters:

```
/home/ubuntu/software/GFAse/build/phase_contacts_with_monte_carlo \
-i porec_to_assembly.sorted_by_read.bam \
-g Assembly-Phased.gfa \
-o /path/to/output/directory/ \
-m 3 \
-t 62
```

Verkko assemblies need the parameters “--skip_unzip” and “--use_homology” in addition to the above parameters, because bubbles are not labeled, and its homopolymer decompressed GFA has incorrectly specified overlaps which cannot be unzipped and stitched trivially. A map quality minimum of 3 is used for all Verkko assemblies.

Evaluation

Input data was evaluated using an alignment based QC tool called WamBam, which iterates BAMs and produces read identity and read length stats[44]. PoreC statistics were generated via a similar method using the “evaluate_contacts” executable provided in the GFAse repository. Reads were aligned to both haplotypes of the trio phased Verkko “full coverage” HG002 assembly, and statistics were calculated by accumulating loci and lengths for each mapping of each read ID. For the “signal ratio” calculation, a contact map was constructed by building a graph similar to that used in phasing methods. Any two mappings of the same read ID constitute an edge, and they are binned by the minimum mapping quality of the pair. Edges that cross from one haplotype to another are considered inconsistent with the true phasing, and are used to compute a ratio of consistent to inconsistent edges.

As a truth set for phasing, chromosome-length haplotypes were generated using Strand-seq and long reads. In order to generate haplotypes, we have used a combination of Strand-seq data and PacBio Hifi reads from the same individual (HG005 and HG002). Sparse and chromosome-length haplotypes were generated using Strand-seq data and R package StrandPhaseR (version 0.99) as previously described[45]. Next, we detected inverted regions using Strand-seq data and manually curated this list of inversions as previously described[46]. We have used this set of inversions to correct Strand-seq phasing over these regions with the StrandPhaseR function called ‘correctInvertedRegionPhasing’ as previously described[46]. After inversion phase correction, we generated dense chromosome-length haplotypes using a combination of Strand-seq haplotypes and PacBio long reads as previously described in the integrative phasing framework[45,47]. Integrative phasing was completed using WhatsHap (version 1.0)[14]. For integrative phasing, we used a defined set of variant positions (available at, https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/ChineseTrio/HG005_NA24631_son/NISTv4.2.1/GRCh38/HG005_GRCh38_1_22_v4.2.1_benchmark.vcf.gz and https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/release/AshkenazimTrio/HG002_NA24385_son/NISTv4.2.1/GRCh38/HG002_GRCh38_1_22_v4.2.1_benchmark.vcf.gz). To include indels into the final callset, we have run WhatsHap with --indels parameter.

To analyze phasing accuracy, a combination of Dipcall[48] and WhatsHap was used. Dipcall is a reference-based variant caller. For a set of phased assemblies, it produces a VCF file of single nucleotide variants (SNVs) as well as small insertions and deletions (INDELs).

For the male HG002 sample, Dipcall was run using the GRCh38 reference, but set to treat the PAR region as autosomal regions. Phase set tags were manually added to the Dipcall VCF file before being used by WhatsHap. WhatsHap "compare" assesses switch error and hamming distance in the phased assemblies by comparing the phasing of alleles in the NIST's Genome In a Bottle[43] truth set to the Dipcall VCF file. WhatsHap "compare" only includes variants in the analysis that have identical alleles in the truth and query VCF file, making it robust to SNVs caused by sequencing errors. WhatsHap "stats" was run with a "chromosme_lengths" input file to calculate phasing statistics.

Collapses and misassemblies within genes were evaluated using minimap2[40], asmgene and the publically available Ensemble genes as input (https://ftp.ensembl.org/pub/release-87/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all.fa.gz). Ensemble cDNA was aligned to the CHM13 v2.0 reference and the HG002 assemblies with minimap2 using "-cx splice:hq" for intra-species cDNA alignment. Asmgene, a part of minimap2, selects the longest isoform from overlapping alignments and counts it as matching the reference if it covers >99% of the transcript length with a mapping identity above an input threshold. To account for ONT's sequencing error profiles, asmgene was run with a mapping identity threshold of 97% instead of 99% used for HiFi assemblies. Transcripts are counted as single-copy if they uniquely align to the reference and multi-copy if they align to multiple loci.

Base quality was estimated using yak[49], based on the k-mer content of Illumina short reads. Each phased assembly was evaluated separately. K-mer in the short reads were counted using "yak count -b 37", and quality values (QV) were estimated using "yak qv -K 3.2g -l 100k". For HG002, we used the 30x Illumina Novaseq PCR-free read set publically available at the Google bucket gs://deepvariant/benchmarking/fastq/wgs_pcr_free/30x/. For the 4 samples from the HPRC (HG01993, HG02132, HG02647, and HG03669), we used 30x Illumina short-reads from the high coverage readset of the 1000 Genomes Project samples[13].

Non-human assemblies (*Noctua fimbriata* and *Sepia bandensis*) were evaluated using default arguments for BUSCO v5.4.3, and the "metazoan_odb10" dataset. To attempt to evaluate the number of phased genes, BUSCO was run twice: once with both haplotypes, and once with one haplotype. For the "both haplotypes" evaluation, the entire diploid assembly was provided to BUSCO. For the "one haplotype" evaluation, one of each haplotype from the phased regions was removed, and the remaining sequences were evaluated by BUSCO.

Switch error in the phased assemblies was also estimated from Illumina short reads from parents. We used yak to count the k-mer in the short reads, as above, and "yak trioeval" to compute the estimated switch error rate. As above, the read sets for HG002's parents were downloaded from the same Google Bucket as for the base quality evaluation, and from the 1000 Genomes project's dataset for the 4 HPRC samples.

Structural variants (SVs) were called from the phased assemblies using dipdiff, a modified version of the SVIM-ASM tool[50]. In HG002 assemblies, the SVs were called against GRCh37 and evaluated with the GIAB SV truthset[32] using Truvari[51]. Truvari's "bench" command was run with "--no-ref a -r 2000 -C 2000" to ignore missing homozygous calls for the reference allele, and match variants up to 2,000 bp away from each other.

References

1. Cordeiro, J. M. *et al.* Compound Heterozygous Mutations P336L and I1660V in the Human Cardiac Sodium Channel Associated With the Brugada Syndrome. *Circulation* **114**, 2026–2033 (2006).
2. Miller, D. B. & Piccolo, S. R. Compound Heterozygous Variants in Pediatric Cancers: A Systematic Review. *Front. Genet.* **11**, (2020).
3. Walker, M. A. *et al.* Novel Compound Heterozygous Mutations Expand the Recognized Phenotypes of FARS2-Linked Disease. *J. Child Neurol.* **31**, 1127–1137 (2016).
4. Marchini, J. & Howie, B. Genotype imputation for genome-wide association studies. *Nat. Rev. Genet.* **11**, 499–511 (2010).
5. Peterson, R. E. *et al.* Genome-wide Association Studies in Ancestrally Diverse Populations: Opportunities, Methods, Pitfalls, and Recommendations. *Cell* **179**, 589–603 (2019).
6. Song, S., Sliwerska, E., Emery, S. & Kidd, J. M. Modeling Human Population Separation History Using Physically Phased Genomes. *Genetics* **205**, 385–395 (2017).
7. Wang, T. *et al.* The Human Pangenome Project: a global resource to map genomic diversity. *Nature* **604**, 437–446 (2022).
8. Chin, C.-S. *et al.* A diploid assembly-based benchmark for variants in the major histocompatibility complex. *Nat. Commun.* **11**, 4794 (2020).
9. Altshuler, D., Donnelly, P., & The International HapMap Consortium. A haplotype map of the human genome. *Nature* **437**, 1299–1320 (2005).
10. Browning, S. R. & Browning, B. L. Rapid and Accurate Haplotype Phasing and Missing-Data Inference for Whole-Genome Association Studies By Use of Localized Haplotype Clustering. *Am. J. Hum. Genet.* **81**, 1084–1097 (2007).

11. Browning, S. R. & Browning, B. L. Haplotype phasing: existing methods and new developments. *Nat. Rev. Genet.* **12**, 703–714 (2011).
12. Howie, B. N., Donnelly, P. & Marchini, J. A Flexible and Accurate Genotype Imputation Method for the Next Generation of Genome-Wide Association Studies. *PLOS Genet.* **5**, e1000529 (2009).
13. Byrska-Bishop, M. *et al.* High-coverage whole-genome sequencing of the expanded 1000 Genomes Project cohort including 602 trios. *Cell* **185**, 3426-3440.e19 (2022).
14. Patterson, M. *et al.* WhatsHap: Weighted Haplotype Assembly for Future-Generation Sequencing Reads. *J. Comput. Biol.* **22**, 498–509 (2015).
15. Edge, P., Bafna, V. & Bansal, V. HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome Res.* **27**, 801–812 (2017).
16. Ebler, J., Haukness, M., Pesout, T., Marschall, T. & Paten, B. Haplotype-aware diplotyping from noisy long reads. *Genome Biol.* **20**, 116 (2019).
17. Koren, S. *et al.* De novo assembly of haplotype-resolved genomes with trio binning. *Nat. Biotechnol.* **36**, 1174–1182 (2018).
18. Cheng, H., Concepcion, G. T., Feng, X., Zhang, H. & Li, H. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nat. Methods* **18**, 170–175 (2021).
19. Rautiainen, M. *et al.* Verkko: telomere-to-telomere assembly of diploid chromosomes. 2022.06.24.497523 Preprint at https://doi.org/10.1101/2022.06.24.497523 (2022).
20. Brandt, D. Y. C. *et al.* Mapping Bias Overestimates Reference Allele Frequencies at the HLA Genes in the 1000 Genomes Project Phase I Data. *G3 GenesGenomesGenetics* **5**, 931–941 (2015).
21. Günther, T. & Nettelblad, C. The presence and impact of reference bias on population genomic studies of prehistoric human populations. *PLOS Genet.* **15**, e1008302 (2019).

22. Wenger, A. M. *et al.* Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nat. Biotechnol.* **37**, 1155–1162 (2019).
23. Luo, X., Kang, X. & Schönhuth, A. phasebook: haplotype-aware de novo assembly of diploid genomes from long reads. *Genome Biol.* **22**, 299 (2021).
24. Duan, H. *et al.* Physical separation of haplotypes in dikaryons allows benchmarking of phasing accuracy in Nanopore and HiFi assemblies with Hi-C data. *Genome Biol.* **23**, 84 (2022).
25. Selvaraj, S., R Dixon, J., Bansal, V. & Ren, B. Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing. *Nat. Biotechnol.* **31**, 1111–1118 (2013).
26. Lieberman-Aiden, E. *et al.* Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. *Science* **326**, 289–293 (2009).
27. Deshpande, A. S. *et al.* Identifying synergistic high-order 3D chromatin conformations from genome-scale nanopore concatemer sequencing. *Nat. Biotechnol.* **40**, 1488–1499 (2022).
28. Payne, A., Holmes, N., Rakyan, V. & Loose, M. BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files. *Bioinformatics* **35**, 2193–2198 (2019).
29. Burton, J. N. *et al.* Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nat. Biotechnol.* **31**, 1119–1125 (2013).
30. Putnam, N. H. *et al.* Chromosome-scale shotgun assembly using an in vitro method for long-range linkage. *Genome Res.* **26**, 342–350 (2016).
31. Kolmogorov, M. Hapdup. (2022).
32. Zook, J. M. *et al.* A robust benchmark for detection of germline large deletions and insertions. *Nat. Biotechnol.* **38**, 1347–1355 (2020).
33. Kolmogorov, M. *et al.* Scalable Nanopore sequencing of human genomes provides a

comprehensive view of haplotype-resolved variation and methylation. 2023.01.12.523790 Preprint at https://doi.org/10.1101/2023.01.12.523790 (2023).

34. The genome sequence of the broad-bordered ... | Wellcome Open Research. https://wellcomeopenresearch.org/articles/6-345.
35. Shafin, K. *et al.* Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nat. Biotechnol.* **38**, 1044–1053 (2020).
36. Deamer, D., Akeson, M. & Branton, D. Three decades of nanopore sequencing. *Nat. Biotechnol.* **34**, 518–524 (2016).
37. Broder, A. Z. On the resemblance and containment of documents. in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)* 21–29 (1997). doi:10.1109/SEQUEN.1997.666900.
38. Carnevali, P. Shasta Methods. *Shasta methods* https://paoloshasta.github.io/shasta/ComputationalMethods.html.
39. Bandage: interactive visualization of de novo genome assemblies | Bioinformatics | Oxford Academic. https://academic.oup.com/bioinformatics/article/31/20/3350/196114.
40. Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**, 3094–3100 (2018).
41. Kokot, M., Dlugosz, M. & Deorowicz, S. KMC 3: counting and manipulating k-mer statistics. *Bioinforma. Oxf. Engl.* **33**, 2759–2761 (2017).
42. Eizenga, J. M. *et al.* Efficient dynamic variation graphs. *Bioinformatics* **36**, 5139–5144 (2020).
43. Zook, J. M. *et al.* Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Sci. Data* **3**, 160025 (2016).
44. nanoporegenomics/wambam: Quick bam QC.

https://github.com/nanoporegenomics/wambam/tree/main.

45. Porubsky, D. *et al.* Dense and accurate whole-chromosome haplotyping of individual genomes. *Nat. Commun.* **8**, 1293 (2017).
46. Porubsky, D. *et al.* Recurrent inversion polymorphisms in humans associate with genetic instability and genomic disorders. *Cell* **185**, 1986-2005.e26 (2022).
47. Hanlon, V. C. T., Porubsky, D. & Lansdorp, P. M. Chromosome-Length Haplotypes with StrandPhaseR and Strand-seq. *Methods Mol. Biol. Clifton NJ* **2590**, 183–200 (2023).
48. Li, H. *et al.* A synthetic-diploid benchmark for accurate variant-calling evaluation. *Nat. Methods* **15**, 595–597 (2018).
49. Li, H. lh3/yak. (2022).
50. Heller, D. & Vingron, M. SVIM-asm: structural variant detection from haploid and diploid genome assemblies. *Bioinformatics* **36**, 5519–5521 (2020).
51. English, A. C., Menon, V. K., Gibbs, R., Metcalf, G. A. & Sedlazeck, F. J. Truvari: Refined Structural Variant Comparison Preserves Allelic Diversity. 2022.02.21.481353 Preprint at https://doi.org/10.1101/2022.02.21.481353 (2022).

Acknowledgements

This work was funded in part by the National Institutes of Health under award numbers: R01HG010485, U24HG010262, U24HG011853, OT3HL142481, U01HG010961, and OT2OD033761. A portion of the R10 HG002 dataset labeled “UUL” in this work was supported by startup funds (Miten Jain, Genome Technology Laboratory, Northeastern University). Cuttlefish Nanopore work was supported by Oxford Nanopore Technologies grant SC20130149 (awarded to Mark Akeson, UCSC Nanopore Group). We acknowledge the support of Oxford Nanopore Technologies staff in generating this data set, in particular the pore-C data.

Declarations

Competing interests

S. J. is an employee of Oxford Nanopore Technologies Inc and a shareholder and/or share option holder of Oxford Nanopore Technologies plc
S. K. M. is an employee of Oxford Nanopore Technologies Inc and a shareholder and/or share option holder of Oxford Nanopore Technologies plc
S. K. has received travel funds to speak at events hosted by Oxford Nanopore Technologies.
E.E.E. is a scientific advisory board (SAB) member of Variant Bio, Inc.
P. C. was an employee of Chan Zuckerberg Initiative during the time most of this work was performed.
K.H.M. is a scientific advisory board (SAB) member of Centaura, Inc.; K.H.M. has received travel funds to speak at events hosted by Oxford Nanopore Technologies.

Availability of Data and Materials

All novel DNA sequence data and genome assemblies will be deposited according to the Genome Standards Consortium (GSC) guidance.

Shasta source code is located at https://github.com/paoloshasta/shasta
GFAse source code is located at https://github.com/rlorigro/GFAse

Bibliography

1. Avery, O. T., Macleod, C. M. & McCarty, M. STUDIES ON THE CHEMICAL NATURE OF THE SUBSTANCE INDUCING TRANSFORMATION OF PNEUMOCOCCAL TYPES : INDUCTION OF TRANSFORMATION BY A DESOXYRIBONUCLEIC ACID FRACTION ISOLATED FROM PNEUMOCOCCUS TYPE III. *J. Exp. Med.* **79**, 137–158 (1944).
2. Ingram, V. M. A Specific Chemical Difference Between the Globins of Normal Human and Sickle-Cell Anæmia Hæmoglobin. *Nature* **178**, 792–794 (1956).
3. Brenner, S., Jacob, F. & Meselson, M. An Unstable Intermediate Carrying Information from Genes to Ribosomes for Protein Synthesis. *Nature* **190**, 576–581 (1961).
4. Central Dogma of Molecular Biology | Nature. https://www.nature.com/articles/227561a0.
5. Gusella, J. F. *et al.* A polymorphic DNA marker genetically linked to Huntington's disease. *Nature* **306**, 234–238 (1983).
6. Sanger, F., Nicklen, S. & Coulson, A. R. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. U. S. A.* **74**, 5463–5467 (1977).
7. Nyrén, P. Enzymatic method for continuous monitoring of DNA polymerase activity. *Anal. Biochem.* **167**, 235–238 (1987).
8. Voelkerding, K. V., Dames, S. A. & Durtschi, J. D. Next-generation sequencing: from basic research to diagnostics. *Clin. Chem.* **55**, 641–658 (2009).
9. Quail, M. A. *et al.* A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics* **13**, 341 (2012).
10. Deamer, D., Akeson, M. & Branton, D. Three decades of nanopore sequencing. *Nat. Biotechnol.* **34**, 518–524 (2016).
11. van Dijk, E. L., Auger, H., Jaszczyszyn, Y. & Thermes, C. Ten years of next-generation sequencing technology. *Trends Genet. TIG* **30**, 418–426 (2014).
12. Levene, M. J. *et al.* Zero-mode waveguides for single-molecule analysis at high concentrations. *Science* **299**, 682–686 (2003).
13. Eid, J. *et al.* Real-time DNA sequencing from single polymerase molecules. *Science* **323**, 133–138 (2009).
14. Wenger, A. M. *et al.* Accurate circular consensus long-read sequencing improves variant

detection and assembly of a human genome. *Nat. Biotechnol.* **37**, 1155–1162 (2019).

15. Payne, A., Holmes, N., Rakyan, V. & Loose, M. BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files. *Bioinformatics* **35**, 2193–2198 (2019).
16. Li, H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. Preprint at https://doi.org/10.48550/arXiv.1303.3997 (2013).
17. Hickey, G. *et al.* Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome Biol.* **21**, 35 (2020).
18. Rausch, T. *et al.* DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics* **28**, i333–i339 (2012).
19. Marks, P. *et al.* Resolving the full spectrum of human genome variation using Linked-Reads. *Genome Res.* **29**, 635–645 (2019).
20. Mostovoy, Y. *et al.* A hybrid approach for de novo human genome sequence assembly and phasing. *Nat. Methods* **13**, 587–590 (2016).
21. Lieberman-Aiden, E. *et al.* Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome. *Science* **326**, 289–293 (2009).
22. Selvaraj, S., R Dixon, J., Bansal, V. & Ren, B. Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing. *Nat. Biotechnol.* **31**, 1111–1118 (2013).
23. Simpson, J. T. & Pop, M. The Theory and Practice of Genome Sequence Assembly. *Annu. Rev. Genomics Hum. Genet.* **16**, 153–172 (2015).
24. Zerbino, D. R. & Birney, E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* (2008) doi:10.1101/gr.074492.107.
25. Pevzner, P. A. & Tang, H. Fragment assembly with double-barreled data. *Bioinforma. Oxf. Engl.* **17 Suppl 1**, S225-33 (2001).
26. Pevzner, P. A., Tang, H. & Waterman, M. S. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.* **98**, 9748–9753 (2001).
27. Rautiainen, M. *et al.* Verkko: telomere-to-telomere assembly of diploid chromosomes. 2022.06.24.497523 Preprint at https://doi.org/10.1101/2022.06.24.497523 (2022).
28. Broder, A. Z. On the resemblance and containment of documents. in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)* 21–29 (1997). doi:10.1109/SEQUEN.1997.666900.
29. Koren, S. *et al.* Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res.* **27**, 722–736 (2017).

30. Shafin, K. *et al.* Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nat. Biotechnol.* **38**, 1044–1053 (2020).
31. Sutton, G. G., White, O., Adams, M. D. & Kerlavage, A. R. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Sci. Technol.* **1**, 9–19 (1995).
32. Myers, E. W. *et al.* A Whole-Genome Assembly of Drosophila. *Science* **287**, 2196–2204 (2000).
33. Wang, T. *et al.* The Human Pangenome Project: a global resource to map genomic diversity. *Nature* **604**, 437–446 (2022).
34. Nurk, S. *et al.* The complete sequence of a human genome. *Science* **376**, 44–53 (2022).
35. Liehr, T. Repetitive Elements in Humans. *Int. J. Mol. Sci.* **22**, 2072 (2021).
36. Fiddes, I. T. *et al.* Human-Specific NOTCH2NL Genes Affect Notch Signaling and Cortical Neurogenesis. *Cell* **173**, 1356-1369.e22 (2018).
37. Auton, A. *et al.* A global reference for human genetic variation. *Nature* **526**, 68–74 (2015).
38. Uffelmann, E. *et al.* Genome-wide association studies. *Nat. Rev. Methods Primer* **1**, 1–21 (2021).
39. Zook, J. M. *et al.* Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Sci. Data* **3**, 160025 (2016).
40. Alkan, C., Coe, B. P. & Eichler, E. E. Genome structural variation discovery and genotyping. *Nat. Rev. Genet.* **12**, 363–376 (2011).
41. Iwata, T. *et al.* A long-range cis-regulatory element for class I odorant receptor genes. *Nat. Commun.* **8**, 885 (2017).
42. Marchini, J. & Howie, B. Genotype imputation for genome-wide association studies. *Nat. Rev. Genet.* **11**, 499–511 (2010).
43. Peterson, R. E. *et al.* Genome-wide Association Studies in Ancestrally Diverse Populations: Opportunities, Methods, Pitfalls, and Recommendations. *Cell* **179**, 589–603 (2019).
44. Song, S., Sliwerska, E., Emery, S. & Kidd, J. M. Modeling Human Population Separation History Using Physically Phased Genomes. *Genetics* **205**, 385–395 (2017).
45. Patterson, M. *et al.* WhatsHap: Weighted Haplotype Assembly for Future-Generation Sequencing Reads. *J. Comput. Biol.* **22**, 498–509 (2015).

46. Edge, P., Bafna, V. & Bansal, V. HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome Res.* **27**, 801–812 (2017).