Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation

Sergey Koren[1*], Brian P. Walenz[1*], Konstantin Berlin[2], Jason R. Miller[3], Adam M. Phillippy[1†]

[1] Genome Informatics Section, Computational and Statistical Genomics Branch, National Human Genome Research Institute, National Institutes of Health, Bethesda, MD USA

[2] Invincea Labs, Arlington, VA USA

[3] J. Craig Venter Institute, Rockville, MD USA

[*] These authors contributed equally to this work

[†] Corresponding author: adam.phillippy@nih.gov

Keywords: de novo assembly, single-molecule sequencing, nanopore sequencing

Abstract

Long-read single-molecule sequencing has revolutionized *de novo* genome assembly and enabled the automated reconstruction of reference-quality genomes. However, given the relatively high error rates of such technologies, efficient and accurate assembly of large repeats and closely related haplotypes remains challenging. We address these issues with Canu, a complete reworking of Celera Assembler that is specifically designed for noisy single-molecule sequences. Canu introduces support for nanopore sequencing, halves depth-of-coverage requirements, and improves assembly continuity while simultaneously reducing runtime by an order of magnitude on large genomes. These advances result from new overlapping and assembly algorithms, including an adaptive overlapping strategy based on *tf-idf* weighted MinHash and a sparse assembly graph construction that avoids collapsing diverged repeats and haplotypes. We demonstrate that Canu can reliably assemble complete microbial genomes and near-complete eukaryotic chromosomes using either PacBio or Oxford Nanopore technologies, and achieves a contig NG50 of greater than 21 Mbp on both human and *Drosophila melanogaster* PacBio datasets. For assembly structures that cannot be linearly represented, Canu provides graph-based assembly outputs for analysis or integration with complementary phasing and scaffolding techniques. Canu source code and pre-compiled binaries are freely available under a GPLv2 license from https://github.com/marbl/canu.

Introduction

The goal of genome assembly is to reconstruct a complete genome from many comparatively short sequencing reads. Overlapping reads that originate from the same region of the genome can be joined together to form contigs, but genomic repeats longer than the overlap length lead to ambiguous reconstructions and fragment the assembly (Phillippy et al. 2008; Nagarajan and Pop 2009). There are two strategies for overcoming this fundamental limitation—increasing the effective read length, and separating non-exact repeats based on copy-specific variations. Recently, single-molecule sequencing has revolutionized assembly by producing reads longer than 10 kbp (Gordon et al. 2016), which has significantly reduced the number of unresolvable repeats (Koren et al. 2012) and enabled the complete assembly of microbial genomes (Chin et al. 2013; Koren et al. 2013; Koren and Phillippy 2014). These long reads also aid assembly phasing (Chin et al. 2016), where the conserved alleles in a diploid, polyploid, or meta- genome can be thought of as a special kind of repeat. However, in contrast to improved read length, single-molecule sequencing is less accurate than past technologies (Eid et al. 2009; Schneider and Dekker 2012), requiring sensitive alignment methods and limiting the discrimination of divergent alleles and non-exact repeats. Nevertheless, PacBio single-molecule real-time (SMRT) sequencing exhibits a largely unbiased and random error model (Ross et al. 2013), enabling assemblies that exceed short-read data both in terms of quality and continuity (Chin et al. 2013; Koren et al. 2013). Oxford Nanopore strand sequencing can also produce highly continuous assemblies, but current biases in base calling prohibit an accurate consensus sequence without the addition of complementary data (Loman et al. 2015).

The increased read length and error rate of single-molecule sequencing has challenged genome assembly programs originally designed for shorter, highly accurate reads. Several new approaches have been developed to address this, roughly categorized as hybrid, hierarchical, or direct (see (Koren and Phillippy 2014) for a review). Hybrid methods use single-molecule reads to reconstruct the long-range structure of the genome, but rely on complementary short reads for accurate base calls (Koren et al. 2012; Hackl et al. 2014; Lee et al. 2014; Salmela and Rivals 2014; Ye et al. 2014; Antipov et al. 2016). Hierarchical methods do not require a secondary technology and instead use multiple rounds of read overlapping (alignment) and correction to

improve the quality of the single-molecule reads prior to assembly (Chin et al. 2013; Koren et al. 2013). Finally, direct methods attempt to assemble single-molecule reads from a single overlapping step without any prior correction (Li 2016; Tørresen et al. 2016). All three approaches are capable of producing an accurate final assembly. However, our goal is the complete reconstruction of entire genomes, so we focus here on the hierarchical strategy because it has produced the most continuous *de novo* assemblies to date (Berlin et al. 2015; Chakraborty et al. 2016).

Canu is a new single-molecule sequence assembler that improves upon and supersedes the now unsupported Celera Assembler (Myers et al. 2000; Miller et al. 2008). Recently, we introduced the MinHash Alignment Process (MHAP) to overcome the computational bottleneck of overlapping noisy, single-molecule sequencing reads (Berlin et al. 2015). Combining this technique with PBcR (Koren et al. 2012) and Celera Assembler, we demonstrated near-complete eukaryotic assemblies from PacBio sequencing alone (Berlin et al. 2015). Building on this work, we developed Canu to (1) integrate our methods into a single, comprehensive assembler, (2) support both PacBio and Oxford Nanopore data, (3) lower runtime and coverage requirements, and (4) improve repeat and haplotype separation. As a result, Canu improves runtime by an order of magnitude for mammalian genomes and outperforms hybrid methods with as little as 20X single-molecule coverage. At higher coverage, reference-quality *de novo* assemblies are possible, including the complete assembly of euchromatic chromosomes from either PacBio or Nanopore sequencing. In addition, Canu's improved graph construction algorithm separates closely related repeats and alleles based on a statistical model of read error, which will be important for future work on diploid, polypoloid, and metagenomic assembly.

Results

Architecture

Canu represents a complete rebuild of the Celera Assembler, shrinking the code base by ~70% and reworking the remaining components to improve usability and performance on single-molecule sequence data. Compared to past releases of Celera Assembler, Canu adds several novel features including computational resource discovery, adaptive k-mer weighting, automated error rate estimation, sparse graph construction, and graphical fragment assembly (GFA) (Li

2016) outputs. The Canu pipeline consists of three stages—correction, trimming, and assembly (Figure 1) —each of which can run independently or in series (e.g. only read correction, or assembly without correction, etc.). When running in a parallel environment, Canu will auto-detect available resources and configure itself to maximize resource utilization. It is currently the most efficient single-molecule read assembler available for large genomes, requiring approximately 20,000 CPU hours to assemble a human genome, compared to ~60,000 required for Falcon (Chin et al. 2016) and >250,000 required for Celera Assembler v8.2 (Berlin et al. 2015). In addition to these runtime improvements, the resulting assemblies are significantly more continuous than prior versions. Details of the improved assembly algorithms and results are given below.

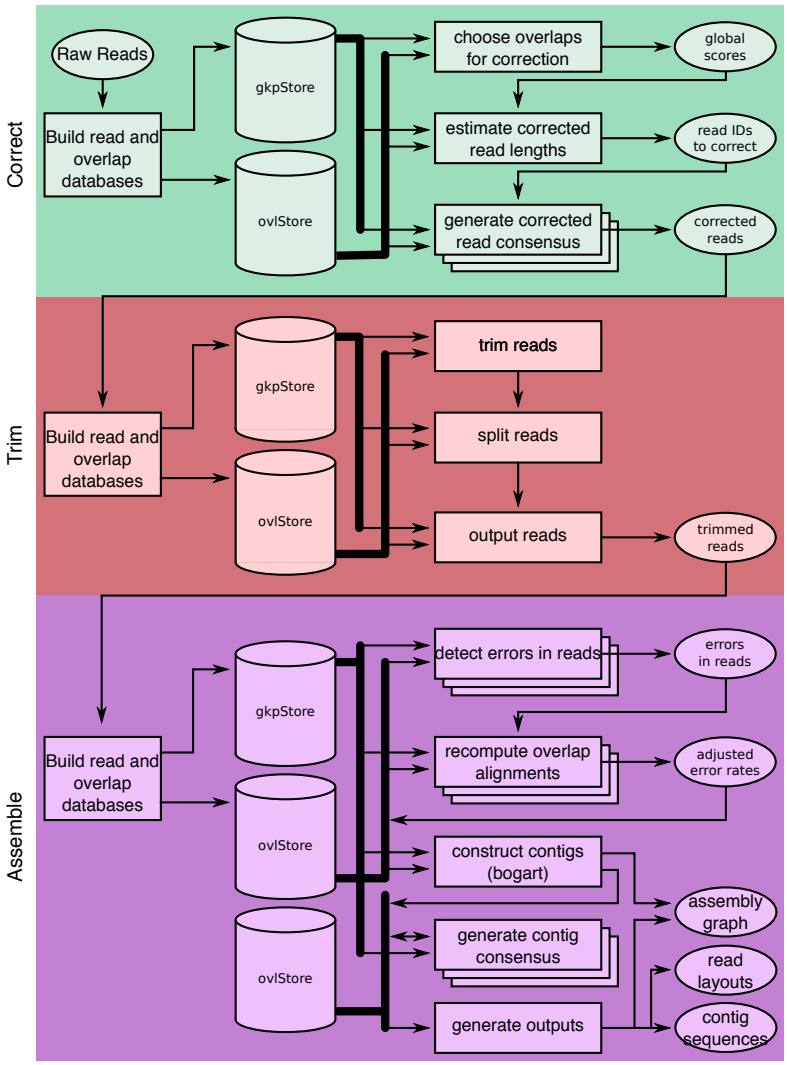

Figure 1. A full Canu run includes three stages: correction (green), trimming (red), and assembly (purple). Canu stages share an interface for binary on-disk stores (databases) as well as parallel store construction. In all stages, the first step constructs an indexed store of input sequences, generates a k-mer histogram, constructs an indexed store of all-vs-all overlaps, and collates summary statistics. The correction stage (green) selects the best overlaps to use for correction, estimates corrected read lengths, and generates corrected reads. The trimming stage (red) identifies unsupported regions in the input and trims or splits reads to their longest supported range. The assembly stage (purple) makes a final pass to identify sequencing errors; constructs the best overlap graph; and outputs contigs, an assembly graph, and summary statistics.

Adaptive MinHash k-mer weighting

Optimal handling of repeats is a challenge, because in addition to fragmenting assemblies, repeats also cause computational bottlenecks during overlapping. Read overlapping typically proceeds in two stages, first building a list of read pairs that share some similarity, and then performing a more direct comparison of those read pairs (e.g. dynamic programming) (Sutton et al. 1995). Candidate overlaps are typically found in the first stage by identifying shared k-mers (length *k* substrings) between all pairs of reads. However, repeats reduce the entropy of the k-mer distribution, and the frequent occurrence of some k-mers significantly increases the number of candidate overlaps that must be processed by the more expensive second stage. A common solution is to mask low-complexity sequence or ignore highly repetitive k-mers during indexing (Ning et al. 2001), as is done by many assemblers including Celera Assembler (Myers et al. 2000), Falcon (Chin et al. 2016), and Miniasm (Li 2016). However, depending on how many repeating k-mers are ignored, some fraction of correct overlaps will not be detected.

Canu takes a more resilient approach to handling repeats that probabilistically reduces, but does not eliminate, the chance a repetitive k-mer will be selected for overlapping. This weighting is achieved via a MinHash overlapping strategy. Rather than comparing individual k-mers to identify potential read overlaps, Canu uses the previously described MinHash Alignment Process (MHAP) to compare compressed sketches of entire reads (Berlin et al. 2015). Because each MinHash sketch contains a fixed-size subset of k-mers selected from a read, the probability of including particular k-mers in a sketch can be adjusted. For instance, a repetitive k-mer occurring many times throughout the genome should have a reduced weight, because it carries relatively little information regarding the origin of the read. In contrast, a relatively unique k-mer occurring multiple times in a single read should have an increased weight, because it represents a larger fraction of the read's length. The combination of these terms represents the relative importance of a k-mer, and in natural language processing this is known as a *tf-idf* weight (term frequency, inverse document frequency).

Application of *tf-idf* weighting to MinHash sketches is straightforward (Chum et al. 2008). Applied to the read overlapping problem, the weighting is a multiplicative combination of the number of occurrences of a k-mer inside a read (the document) and the overall popularity of the k-mer among all reads (the corpus). For document similarity, the intuition is that a rare word

that occurs multiple times in a single document is a good candidate to identify similar documents. For read overlapping, this statistic has the desirable property that repetitive k-mers receive low weights. By reducing the occurrence of repetitive k-mers within sketches, the distribution of indexed k-mers becomes more uniform. This reduces the number of uninformative, repetitive overlaps that are identified during sketch comparison, significantly improving both runtime and memory usage. Importantly, this is achieved via a probabilistic process so no repeat masking is required and true overlaps between repetitive reads will still be recovered. Alternative weighting schemes are also possible with this technique (e.g. to increase the probability of selecting haplotype-specific k-mers), but we focus our evaluation on the *tf-idf* statistic.

We evaluated *tf-idf* weighting on a *Bacillus anthracis* genome sequenced with the Oxford Nanopore MinION (Supplementary Note 1, 2). The *B. anthracis* Sterne strain makes a useful test because it possesses a single plasmid often present in multiple copies relative to the main chromosome. In this case, the pXO1 plasmid presented at approximately 6-fold higher coverage than the chromosome (487X vs. 76X). This variable sequencing depth challenges traditional k-mer filtering strategies based on a fixed, all-or-nothing threshold. Additionally, it is critically important to recover such plasmids during sequencing, because increased copy number has been previously associated with virulence in other species like *Yersinia pestis* (Wang et al. 2016). As expected, MHAP overlap sensitivity for the plasmid is low (26%) when repetitive k-mers are filtered via a fixed threshold. Similarly low sensitivity is seen from Minimap (Li 2016) and DALIGNER (Myers 2014), which both employ a k-mer count threshold by default (17% and 60%, respectively, Supplementary Table S1). Manually increasing this threshold to include plasmid k-mers improves Minimap and DALIGNER sensitivity to 94% and 76%, respectively. However, Minimap suffers a drop in positive predictive value (PPV), reporting more false, repeat-induced overlaps. DALIGNER performs a dynamic programing check to confirm all candidate overlaps, so its PPV remains high, but it suffers both a memory (1.6-fold) and runtime (2-fold) penalty. In contrast, Canu's adaptive *tf-idf* weighting scheme requires no parameter adjustment and achieves 89% sensitivity and maintains high PPV (99.5%) with no added runtime or memory penalty.

Best overlap graph

Canu uses a variant of the greedy "best overlap graph" (BOG) algorithm from (Miller et al. 2008) for constructing a sparse overlap graph. Loading the full overlap graph into memory, as required by string graph formulations (Myers 2005), can be costly for large, complex genomes. In contrast, the greedy algorithm loads only the "best" (longest) overlaps for each read end into memory. This greedy approach is optimal when the read length is sufficiently long (Bresler et al. 2013), and a best overlap graph can be built using just 64 GB of memory for a mammalian genome. However, the greedy algorithm can be misled by repeats that are longer than the overlap length and is therefore prone to mis-assemblies. Canu's new "Bogart" algorithm addresses this problem by statistically filtering repeat-induced overlaps and retrospectively inspecting the graph for potential errors.

In the original BOG method, the best overlaps were selected from a pool of all overlaps below a user-specified error rate threshold, where the overlap error rate is defined as the edit distance divided by the length of the overlap alignment. Thus, this threshold must be set low enough that repeats do not result in false overlaps, yet high enough to account for sequencing error and detect true overlaps. In the new Bogart method, the optimal overlap error rate parameter is automatically estimated from the data, both globally and locally. However, this presents a challenge for raw single-molecule data, which has a sequencing error rate between 10–20% that blurs the distinction between noise and repeat-induced overlaps. Therefore, Canu performs multiple rounds of read and overlap error correction prior to graph construction. After these corrections, the residual read error is estimated from the distribution of all longest overlaps. This full overlap set is then filtered to include only those overlaps within some tolerance of the global median error rate (Figure 2a), and the longest overlaps are recomputed using only this subset. Compared to prior versions of BOG that used a 5% default overlap error rate, Bogart will typically discover an overlap error rate below 2% for corrected single-molecule data. This low threshold effectively removes most false overlaps, allowing the greedy method to construct a clean best overlap graph. From this graph, initial contigs are constructed from the maximal non-branching paths.

Despite careful correction and overlap filtering, exact or near exact repeats within the error rate tolerance can still add false edges to the graph, resulting in potential mis-assemblies that incorrectly join distant parts of the genome. To guard against this, each initial contig is

inspected to identify and correct potential errors. First, the expected overlap error rate for each position of the contig is locally computed using the best overlaps (Figure 2b). Next, all non-best overlaps to reads outside the contig within some deviation of the expected error rate are collected. This excludes sufficiently diverged repeats and haplotypes, while the retaining overlaps that are compatible with the local error profile. These overlaps are used to annotate potential alternative branches within the contig and flagged for further inspection. If a branching region is spanned by at least one read (Figure 2c) (Ukkonen 1992) or there is no alternate overlap of similar quality (Figure 2d), it is confirmed as correct. Otherwise, the region is split into at least three new contigs and labeled as an unresolved repeat.

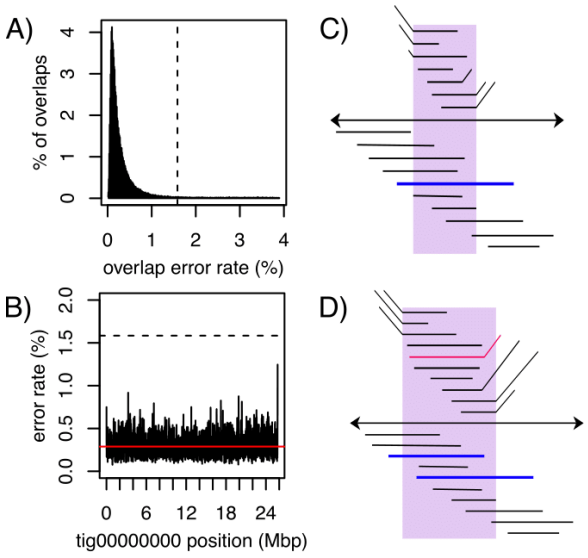

Figure 2. An illustration of overlap error rate estimation, repeat identification, and splitting. A) A histogram of all best edge error rates with the auto-selected threshold shown as a dashed line for the *D. melanogaster* PacBio dataset. All overlaps up to 4% error were computed. However, the peak error rate is at 0.25% error and Canu chose to use only overlaps below 1.6% error for graph construction on this dataset. B) The dashed line shows the global error rate threshold (1.6%), and the profile shows the locally computed error rate for the largest contig in this assembly. Only overlaps consistent with this local error rate are considered as potential alternate paths when supplementing the initial best overlap graph. By adjusting the error rate for each contig, Canu can separate diverged repeats without making an assumption of uniform read error across the assembly. C) The contig is shown as a black line with arrows on both sides, indicating Bogart extends a path in both the 5' and 3' directions until encountering no overlaps or a read that is already incorporated in another contig. Repeat regions annotated by conflicting reads, are shown above the contig. The reads align to part of the contig (the repeat) but indicate a different boundary sequence. A single read (blue line) spans the full repeat region, indicating the contig reconstruction is correct. D) Repeat regions annotated by conflicting reads as before. In this case, no single read spans the full repeat region, and the initial contig was built using the overlap between two blue reads. The contig is split if the overlap between the two blue reads is not significantly better than the overlap from either blue read to the conflicting red read.

After construction and validation, Canu provides a representation of the final assembly graph in the Graphical Fragment Assembly (GFA) format (Li 2016). This representation is equivalent to a sparse read overlap graph, simplified to remove unambiguous paths and contained reads. Figure 3 shows the Canu assembly graph for *Drosophila melanogaster* sequenced using PacBio. Some chromosome arms are assembled into single contigs, but the graph reveals the structure of the more complex, unresolved repeats in the assembly. For

example, chromosome 2L is assembled as a single component in the graph, but is broken towards the end due to a large array of transposable elements and the histone gene cluster, which spans over 500 kbp (Hoskins et al. 2015). These elements also correspond to unfinished gaps in the *D. melanogaster* reference. Canu's graphical output localizes this complex structure to a specific chromosome arm and location. However, the size of the repeats precludes complete assembly. Combining the Canu assembly graph with supplementary long-range information, such as from optical (Hastie et al. 2013) or chromatin contact mapping (Burton et al. 2013; Kaplan and Dekker 2013), could help identify the correct path and resolve such structures.

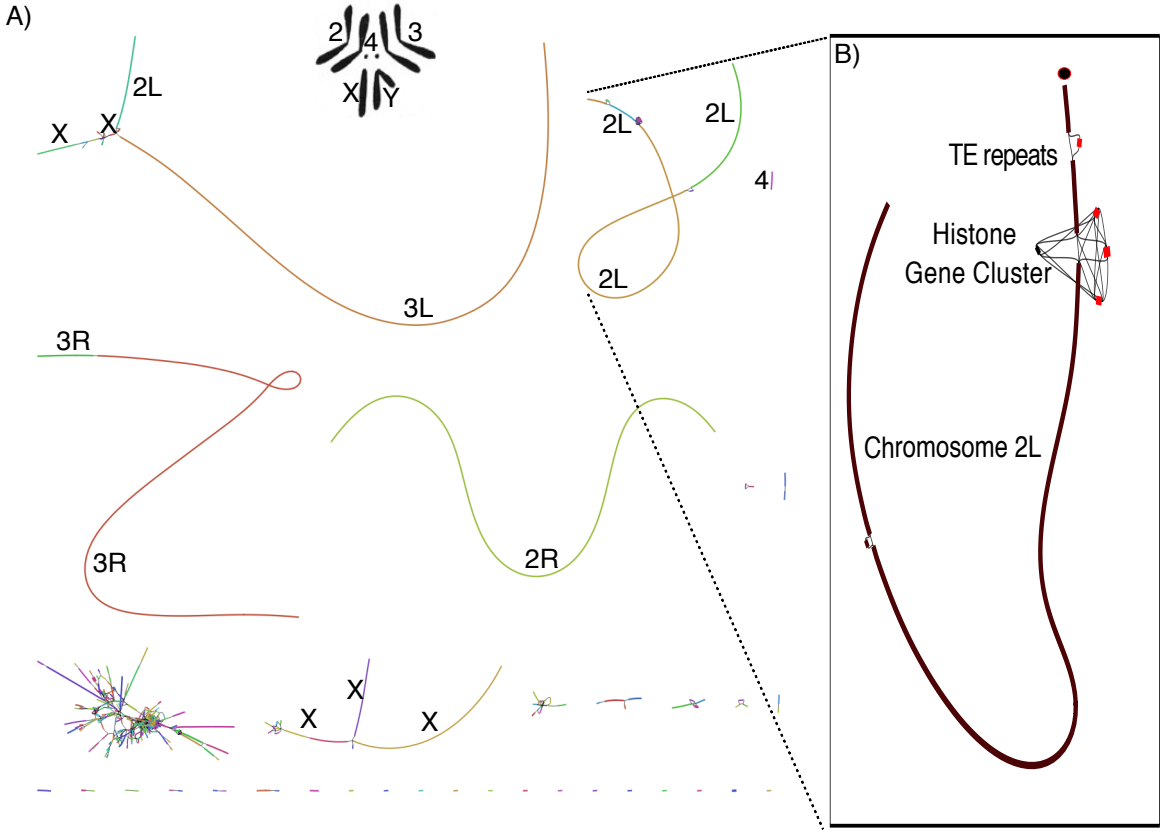

Figure 3: Canu GFA output localizes complex repeat regions, allowing for improved scaffolding. A) Bandage (Wick et al. 2015) plot of *D. melanogaster* compared to the karyotype (Stevens 1912; Metz 1914) from FlyBase (Attrill et al. 2016). Nodes are contigs sized by length and edges indicated unused overlaps between contigs. The largest contigs are colored randomly and labeled with their chromosome based on alignment to the reference. B) The callout shows a subset of chromosome 2L from positions 3.07 Mbp to 23.12 Mbp, reordered with the centromere at the top (indicated by a filled circle). Unique contigs are shaded black while repeat contigs are shaded red. While the 2L chromosome scaffold is composed of 10 individual contigs, they are all linked in the output graph. The two red regions correspond to reference gaps at positions 2L:21,485,538, which consists of 100–200 copies of the histone gene cluster spanning over 500 kbp and 2L:22,420,241 which is bordered by several TE repeats (Hoskins et al. 2015). Even though Canu is unable to fully resolve these large repeat arrays, the graph indicates large-scale continuity across chromosome 2L and could enable resolution with secondary technologies.

Low-coverage hierarchical assembly

Canu substantially lowers the coverage requirements for single-molecule *de novo* assembly. Previously, at least 50X coverage was recommended for hierarchical assembly methods (Berlin et al. 2015; Chakraborty et al. 2016). However, as sequencing lengths and algorithms have improved, so have the minimum input requirements. To quantify performance and determine when a hybrid method may be preferred, we randomly subsampled 10–150X of PacBio P5-C3 coverage from *Arabidopsis thaliana* Ler-0 (Kim et al. 2014) and compared Canu assemblies to both Illumina-only and hybrid assemblies using SPAdes (Antipov et al. 2016). At 20X single-molecule coverage, the Canu assembly is more continuous than the hybrid SPAdes assembly of 20X PacBio combined with 100X Illumina. Although making efficient use of low coverage PacBio data, the hybrid method plateaus after 30X, and the continuity of the Canu 20X assembly is comparable to the best hybrid assembly given 150X of PacBio (Figure 4, Supplementary Note 3, Supplementary Table S2, Supplementary Figure S1). In contrast, Canu continues to improve with increasing PacBio coverage, reaching its maximum assembly continuity around 50X. The amount of improvement is a function of the repeat content and sequence length. PacBio sequence lengths follow a log-normal distribution (Ono et al. 2013), and additional coverage increases the probability of spanning a long repeat. Thus, we would expect continued improvement with higher coverage for larger, more complex genomes. Thus, we recommend the hierarchical method whenever single-molecule coverage exceeds 20X. However, as shown earlier, consensus accuracy from low coverage single-molecule data is limited (Pacific Biosciences 2015), and polishing (Walker et al. 2014) with short reads is recommended after assembly (Supplementary Table S2).

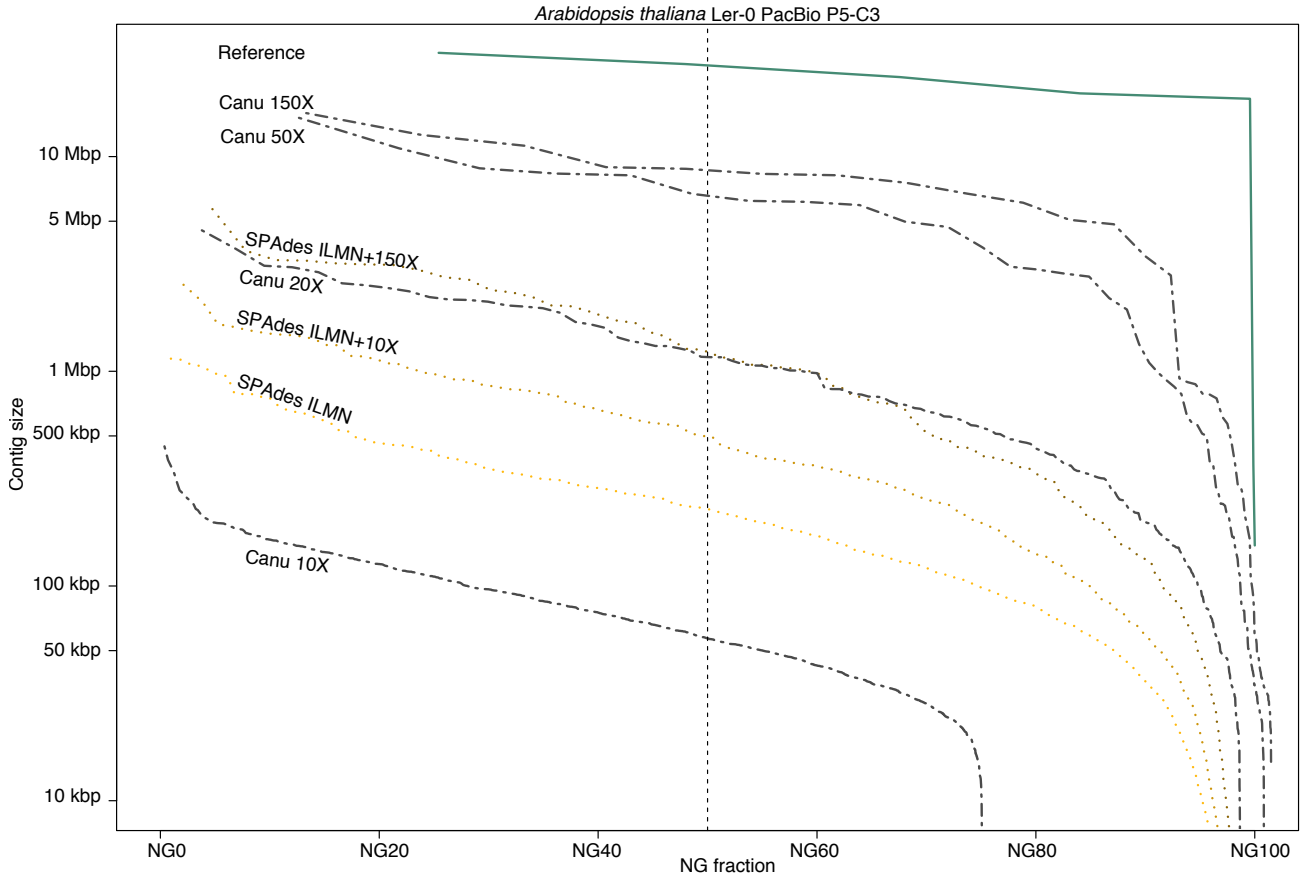

Figure 4: A comparison of assembly continuity between Canu and SPAdes. Each set of contigs is sorted from longest to shortest and plotted versus a cumulative percentage of the genome covered. Assemblies with larger contigs appear in the top of the plot. The ideal assembly corresponds to the green reference line. The commonly used NG50 metric corresponds to the vertical dashed line. Canu quickly gains continuity with increasing coverage, approaching the limit with 50X PacBio on this genome. In contrast, while making a large gain from Illumina-only to 10X PacBio, SPAdes continuity plateaus by 30X, and the Canu 20X assembly is comparable to the hybrid SPAdes assembly using 150X PacBio.

Assembly evaluation

We evaluated Canu on a variety of microbial and eukaryotic genomes, and compared with Falcon (Chin et al. 2016), Miniasm (Li 2016), and hybrid SPAdes (Antipov et al. 2016) using both PacBio and Oxford Nanopore sequencing data (Supplementary Note 2, 4-6). Continuity was measured using maximum and NG50 contig size, where NG50 is the longest contig such that contigs of this length or greater sum to at least the haploid genome size. Accuracy was computed via alignment to the nearest available reference genome using MUMmer (Kurtz et al. 2004), and reported using the GAGE (Salzberg et al. 2012) metrics, which evaluate both base (single nucleotide) and structural (inversions, relocations, and

translocations) errors. An ideal assembly has high continuity, low errors, and high base accuracy, with 99.99% (Phred QV40 (Ewing and Green 1998)) commonly defined as the minimum quality for a "finished" sequence (Felsenfeld et al. 1999; Schmutz et al. 2004).

PacBio sequence assembly

We assembled bacterial and eukaryotic genomes recently released (Kim et al. 2014) and available from PacBio DevNet (PacBio DevNet 2014). Table 1 shows that Canu produces the most continuous assembly on three of the four eukaryotic genomes tested, while maintaining high accuracy (Supplementary Figure S2-S6). In the one case that Miniasm produces a higher NG50 (*Caenorhabditis elegans*), both Falcon and Miniasm introduce large-scale structural rearrangements not present in the Canu assembly (Supplementary Figure S5). For assembly runtime, Miniasm (Li 2016) is an order of magnitude faster than Canu and Falcon (Supplementary Table S3-S6). However, in contrast to Canu and Falcon, Miniasm does not perform a gapped alignment for either overlapping or consensus generation. Instead, Miniasm generates a string graph (Myers 2005) directly from approximate read overlaps and labels the edges of this graph with the raw read sequences. Thus, the average identity of the resulting assembly is equal to the identity of the input sequences, and the approximate overlap positions can leave large insertions and deletions in the assembly. As a result, the Miniasm assemblies have both low base accuracy (<90%) and a higher frequency of large insertions and deletions, which can be difficult to remove during polishing. Therefore, Miniasm requires four rounds of Quiver polishing (Chin et al. 2013) before the assembly quality converges (Supplementary Table S7-S10), whereas Canu requires only a single polishing round and is ultimately fastest to generate a high-quality assembly (Table 1).

Canu shows good scaling to mammalian genomes, completing a polished human assembly threefold faster than Falcon and many times faster than the prior version of Celera Assembler (Supplementary Tables S3-4). Canu runtime improvements come from recent optimizations to the initial overlapping and read correction process (Methods), which have traditionally been the slowest step in hierarchical assembly. Read correction is now the fastest step of the Canu pipeline. As a result, Canu is often able to generate a complete assembly in less time than Falcon requires for its initial DALIGNER (Myers 2014) overlapping stage (Supplementary Table S3-4). On the human genome, where the upfront cost of building MHAP sketches is most effectively amortized, Canu's initial overlapping step is also faster than

Minimap (Supplementary Table S3, S5), but Miniasm failed to assemble this dataset due to its in-memory string graph construction, which exceeded 1 TB of memory. Canu's greedy algorithm required less than 36 GB for the same dataset.

Table 1: Canu is fastest to a polished assembly with high genome completeness and continuity for PacBio sequencing data

Genome	Asm/Polish	Max (Mbp)	NG50 (Mbp)	% Ref	# Errors	Time (CPU h)	% Idy
E. coli	Canu+Quiver	4.68	4.68	100%	0	12.25	>99.99%
	Falcon+Quiver	4.64	4.64	100%	2	25.14	>99.99%
	Miniasm+Quiver	4.64	4.64	99.99%	2	31.93	>99.99%
	SPAdes	4.64	4.64	100%	0	4.09	>99.99%
D. melanogaster	Canu+Quiver	25.78	21.31	97.47%	1,025	1,396.52	99.98%
	Falcon+Quiver	23.08	9.84	96.12%	1,054	2,305.92	99.98%
	Miniasm+Quiver	15.85	5.84	96.51%	752	1,484.33	99.98%
A. thaliana	Canu+Quiver	15.95	8.31	82.94%	220	925.31	99.07%
	Falcon+Quiver	15.94	8.17	82.72%	222	1,132.25	99.07%
	Miniasm+Quiver	11.61	5.07	82.88%	205	976.43	99.07%
C. elegans	Canu+Quiver	5.34	2.35	99.70%	139	410.07	99.97%
	Falcon+Quiver	4.99	1.88	98.82%	138	397.40	99.97%
	Miniasm+Quiver	5.85	2.96	99.44%	141	526.16	99.97%
CHM1	Canu+Quiver	80.08	21.95	86.84%	1,105	22,749.71	99.81%
	Falcon+Quiver	52.34	9.46	86.58%	1,082	68,789.00	99.81%
	Miniasm+Quiver	N/A	N/A	N/A	N/A	N/A	N/A

Genome: the genome being assembled. Asm/Polish: software tools used to generate an initial and polished assembly. Max: the maximum contig size, in Mbp. NG50: N such that 50% of the genome is contained in contigs of length >=N where the genome size is set to the reference length (excluding alternates in Ref38). % Ref: the percentage of the reference covered by assembly alignments; # Errors: GAGE structural differences compared to the reference. Time: total time to generate a finished assembly, including time to polish consensus with Quiver (Chin et al. 2013). % Idy: identity to the reference of the final polished assembly. Multiple rounds of Quiver were run until the identity converged. This translated to a single round for Falcon and Canu and four rounds for Miniasm. Miniasm on CHM1 required over 1.5 TB of memory and could not complete. SPAdes results on *E. coli* are without Quiver, making it faster than polished assemblies. However, the initial SPAdes assembly has similar quality to Canu (QV45 vs QV47 respectively) in equivalent runtimes (4.09 SPAdes vs. 4.26 Canu CPU hours) (Supplementary Table S7, S10). Quiver polishing of the Canu assembly exceeds QV58, beating the best SPAdes polished assembly. Based on SPAdes benchmarking on *A. thaliana* above, it was excluded from eukaryotic runs. *A. thaliana* and CHM1 differ from the reference, leading to lower identity and reference coverage for all assemblers.

Canu also represents a dramatic improvement over the latest version of Celera Assembler (Berlin et al. 2015). Our previous PacBio P5-C3 human (CHM1) assembly required >250,000 CPU hours with Celera Assembler, resulting in a contig NG50 of 4 Mbp (Berlin et al. 2015). The re-assembly of this same dataset with Canu required <25,000 CPU hours and the NG50 increased to over 7 Mbp. Improvements to PacBio chemistries are also resulting in impressive assembly gains. An updated assembly using the more recent PacBio P6-C4 chemistry requires the same runtime, yet increases the NG50 5-fold to over 20 Mbp. This *de novo* Canu assembly has comparable assembly size, contig counts, and continuity to the human reference assemblies

before NCBI Build 34 (ca. 2003), which is the release immediately prior to the "finished" human genome (International Human Genome Sequencing 2004). The contig sizes of this Canu human assembly are also comparable to the scaffold sizes generated by Celera (Istrail et al. 2004), which used Sanger sequencing with a range of insert sizes and BACs.

Nanopore sequence assembly

Currently, the Oxford Nanopore MinION can read either one or both strands of a double-stranded DNA molecule. The "1D" mode sequences only the template strand, whereas the "2D" mode sequences both the template and complement strands via a hairpin adapter. This technique is similar to PacBio circular consensus sequencing (CCS) (Travers et al. 2010). Because the 2D mode provides two independent observations of each base, the per-read accuracy is improved (e.g. from 70% to 86% for R7.3 chemistry (Figure 5a)). To date, all assembly evaluations have focused on the more accurate 2D sequences (Loman et al. 2015; Judge et al. 2016; Sovic et al. 2016). While more accurate, the library preparation for 2D sequencing is more complex, reduces the effective throughput of the instrument (each molecule must be read twice), and currently produces shorter sequences (Oxford Nanopore Technologies 2016b). Thus, we designed Canu to assemble both 2D and the noisier 1D sequences, which benefit from increased read length and throughput, both key factors for genome assembly.

Table 2 shows Canu assemblies of seven recent 2D Nanopore sequencing runs (Loman et al. 2015; Quick and Loman 2015). Consistent with independent evaluations (Judge et al. 2016; Sovic et al. 2016), Canu produces highly continuous assemblies from Nanopore data alone, and the continuity of Canu assemblies was equal to or better than all assemblers tested. Miniasm was again extremely fast and produced structurally correct and contiguous assemblies (Supplementary Fig S7-S13, Supplementary Table S11-S13), except for *B. anthracis*, where it failed to assemble the high-copy plasmid pXO1 due to its stringent k-mer filtering. As with PacBio, the initial Minimap assemblies also have low base accuracy. For Nanopore data, Minimap assemblies were less than 90% accurate, whereas Canu assemblies typically exceeded 99%. Consensus polishing using the Nanopore signal data with Nanopolish (Loman et al. 2015) further improved the accuracy of all assemblies to as high as 99.85%, but polishing the lower quality Miniasm assemblies to comparable accuracy was 750% slower (Supplementary Table S11–S13).

Table 2: Canu consistently assembles complete genomes from only Oxford Nanopore sequencing data

Genome	Asm/Polish	# Contigs	Max (Mbp)	% Ref	# Errors	Time (CPU h)	% Idy
E. coli MAP005	Canu+Nanopolish	(1)	4.64	99.98%	2	376.87	99.43%
	Falcon+Nanopolish	105	0.42	23%	2	106.2	99.41%
	Miniasm+Nanopolish	3	3.40	99.96%	0	2,344.02	99.36%
E. coli MAP006-1	Canu+Nanopolish	(1)	4.63	99.80%	0	167.04	99.81%
	Falcon+Nanopolish	(1)	4.63	99.86%	0	207.45	99.78%
	Miniasm+Nanopolish	(1)	4.66	99.97%	2	1,801.02	99.72%
E. coli MAP006-2	Canu+Nanopolish	(1)	4.64	99.91%	2	168.69	99.78%
	Falcon+Nanopolish	(1)	4.64	99.94%	2	196.16	99.76%
	Miniasm+Nanopolish	(1)	4.65	99.70%	4	1,482.95	99.69%
E. coli MAP006-PCR-1	Canu+Nanopolish	(1)	4.64	99.95%	0	164.08	99.84%
	Falcon+Nanopolish	(1)	4.63	99.80%	2	168.37	99.82%
	Miniasm+Nanopolish	3	2.15	99.96%	0	1,338.28	99.77%
E. coli MAP006-PCR-2	Canu+Nanopolish	(1)	4.64	99.99%	2	206.09	99.85%
	Falcon+Nanopolish	(1)	4.64	100.00%	2	212.89	99.84%
	Miniasm+Nanopolish	(1)	4.65	99.98%	0	1,669.83	99.81%
B. anthracis	Canu+Nanopolish	(2)	5.20	99.77%	0	894.40	99.14%
	Falcon+Nanopolish	31	0.47	86.29%	0	795.93	99.17%
	Miniasm+Nanopolish	4	5.22	97.21%	0	5,094.90	99.05%
Y. pestis CO92	Canu+Nanopolish	(4)	4.67	99.97%	11	254.25	99.76%
	Falcon+Nanopolish	(4)	4.68	99.97%	12	295.01	99.72%
	Miniasm+Nanopolish	9	2.69	99.91%	11	2,000.16	99.65%

Columns defined as in Table 1. Since the maximum contig size is usually the NG50 size of these bacterial genomes, the # of contigs over 2 kbp in length is included to indicate assembly completeness. Genomes where the number of contigs matches the number of organelles in the reference are marked with parentheses, indicating they are complete. Multiple rounds of Nanopolish were run until QV converged. This was one round for Falcon and Canu and three rounds for Miniasm. Nanopolish suffers a large performance penalty on high-error inputs, leading to significantly longer runtimes on initial Miniasm inputs. The *B. anthracis* and *Y. pestis* genome were not the same strain used for validation, leading to higher error counts and lower identity. In the case of *Y. pestis,* all assemblers agreed on three large inversions with respect to the reference (Supplementary Fig S13).

Generating a finished-quality (>99.99%) consensus sequence from Nanopore reads required polishing with complementary short-read data. We repeated the above evaluation, but substituted Pilon (Walker et al. 2014) for Nanopolish (Loman et al. 2015), and included comparisons to hybrid SPAdes (Table 3). Pilon aligns Illumina reads against an assembled sequence and corrects base errors and small insertions and deletions (Indels). As with Nanopolish, this process was iterated until consensus quality converged, except for hybrid SPAdes, which did not require additional polishing. Combined assembly and polishing times for all assemblers were comparable. Canu, Falcon, and SPAdes routinely exceeded 99.99% polished base accuracy, but Miniasm was unable to exceeded 99.9% after many rounds of polishing (Supplementary Table S14). The residual Miniasm errors were large (average >500 bp) expansions or collapses in the draft assembly (Supplementary Figure S14), which are difficult to

correct using short-read sequences. Hybrid SPAdes was typically most accurate, both in terms of base and structural accuracy. However, on the repetitive *Y. pestis* genome, it was significantly less contiguous than hierarchical methods, and on the newer high-quality Nanopore datasets, the polished Canu accuracy exceeded SPAdes (Supplementary Table S15-S18, Supplementary Fig S15-S21).

Table 3. Canu exceeds hybrid methods in continuity while matching identity when polished with Illumina data

Genome	Asm/Polish	# Contigs	Max (Mbp)	% Ref	# Errors	Time (CPU h)	% Idy
E. coli MAP005	Canu+Pilon	(1)	4.65	99.99%	2	10.98	99.99%
	Falcon+Pilon	105	0.42	23.04%	2	4.36	99.95%
	Miniasm+Pilon	3	3.40	90.62%	42	3.15	97.39%
	SPAdes	(1)	4.64	100.00%	0	3.61	>99.99%
E. coli MAP006-1	Canu+Pilon	(1)	4.63	99.82%	0	5.89	>99.99%
	Falcon+Pilon	(1)	4.63	99.86%	0	7.3	>99.99%
	Miniasm+Pilon	(1)	4.66	96.97%	21	3.14	99.61%
	SPAdes	(1)	4.64	100.00%	0	3.65	>99.99%
E. coli MAP006-2	Canu+Pilon	(1)	4.64	99.94%	2	3.92	>99.99%
	Falcon+Pilon	(1)	4.64	99.94%	2	3.93	99.99%
	Miniasm+Pilon	(1)	4.64	97.98%	26	2.73	99.63%
	SPAdes	(1)	4.64	100.0%	0	3.56	>99.99%
E. coli MAP006-PCR-1	Canu+Pilon	(1)	4.64	99.95%	0	4.15	>99.99%
	Falcon+Pilon	(1)	4.63	99.80%	2	3.55	>99.99%
	Miniasm+Pilon	3	2.16	98.41%	12	2.15	99.67%
	SPAdes	2	3.95	100.00%	0	3.56	>99.99%
E. coli MAP006-PCR-2	Canu+Pilon	(1)	4.64	100.00%	2	6.16	>99.99%
	Falcon+Pilon	(1)	4.64	100.00%	2	9.22	>99.99%
	Miniasm+Pilon	(1)	4.65	98.57%	20	2.69	99.67%
	SPAdes	(1)	4.64	100.00%	0	4.00	>99.99%
B. anthracis	Canu+Pilon	(2)	5.21	99.77%	1	65.01	99.85%
	Falcon+Pilon	31	0.48	86.31%	0	14.95	99.89%
	Miniasm+Pilon	4	5.25	79.36%	44	4.9	92.28%
	SPAdes	6	4.13	100.00%	0	8.47	99.99%
Y. pestis CO92	Canu+Pilon	(4)	4.66	99.83%	23	17.92	99.89%
	Falcon+Pilon	(4)	4.64	99.65%	26	10.63	99.87%
	Miniasm+Pilon	9	2.70	93.76%	42	8.68	98.79%
	SPAdes	29	0.37	95.99%	15	17.08	99.96%

Columns defined as in Table 1. Hybrid assembly using Oxford Nanopore and Illumina data was tested across the assemblers from Table 2 with the addition of SPAdes. Polishing on all assemblies, except SPAdes, was done with three rounds of Pilon and total times reported. As in Table 2, Canu is most consistent at producing closed genomes for Oxford Nanopore data. SPAdes runtime is comparabled to polished Canu runtimes with both exceeding 99.99% identity on the majority of genomes. SPAdes has higher identity on the older MAP005 data, *B. anthracis,* and *Y. pestis.* However, Canu polished identities exceed SPAdes identities on the remaining datasets.

Nanopore 1D sequence assembly

We evaluated the performance of Canu on noisy 1D data using only the template sequences from the *Escherichia coli* MAP006-1 dataset (Quick and Loman 2015), which averaged a raw 1D accuracy of just 70% (Figure 5a). To deal with this high error, we exploited the modularity of Canu to run ten rounds of correction, with the output of each round fed as input to the next (Supplementary Note 11). The corrected reads were then assembled into ten contigs with an NG50 of 619 kbp and a maximum contig size of 1.22 Mbp covering 89% of the reference at 85.52% identity versus a single circular chromosome for 2D data (Figure 5b-c). In contrast, the Miniasm assembly of this data covered less than 10% of the reference at 76.76% identity (Supplementary Figure S22). Polishing the Canu assembly with Nanopolish converged on a 1D consensus accuracy of 98% identity, and short-read polishing with Pilon improved the assembly to 93.83% coverage and 99.72% identity. Thus, despite their high error, we conclude that 1D sequences as low as 70% identity can be assembled, albeit at reduced consensus quality. However, more recent Nanopore sequencing chemistries are producing 1D reads with 85% accuracy (Quick and Loman 2016), for which only a single round of correction is necessary.

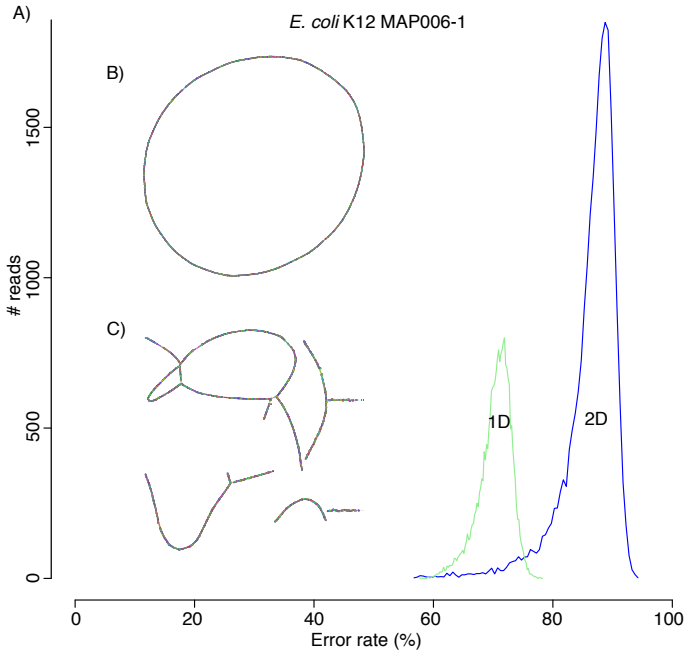

Figure 5: Canu can assemble both 1D and 2D nanopore *E. coli* reads. A) A comparison of error rates for 1D and 2D read error rates versus the reference. Template 1D and 2D reads from the MAP006-1 *E. coli* dataset (Quick and Loman 2015) were aligned independently to compute an identity for all reads with an alignment over 90% of their length (95% of the 2D sequences and 86% of the 1D reads had an alignment over 90% of their length). The 2D sequences averaged 86% identity and the 1D reads averaged 70% identity. B) Bandage plot of the Canu best overlap graph for the 2D data. The genome is in a single circle representing the full chromosome. C) The corresponding plot for 1D data. While highly contiguous, there are multiple components due to missed overlaps and unresolved repeats (due to the higher sequencing error rate).

Few eukaryotic Nanopore datasets are currently available due to the low throughput of the initial MinION instruments. However, as previously demonstrated using PacBio data, Canu easily scales to mammalian-sized genomes, and as Nanopore throughput improves it is expected that highly continuous eukaryotic assemblies will be possible. For an early test, we assembled the *Saccharomyces cerevisiae* genome from available R6 and R7 MinION data (Goodwin et al. 2015). This older dataset contains only 20X coverage of 2D reads and an average identity of 70% (Figure 6a), significantly lower than produced by newer chemistries (Quick and Loman 2015). Despite this, Canu was able to assemble the dataset using the same iterative correction strategy as for 1D reads (Figure 6b, Supplementary Note 12, Supplementary Figure S23). The resulting assembly comprises 41 contigs, with a majority of chromosomes in one or two contigs and an NG50 of 469 kbp covering 95.22% of the reference at 94.33% identity. Illumina polishing with Pilon improved the assembly to 96.86% coverage at 99.83% identity. Prior to Canu, this dataset could only be assembled via a hybrid approach. Newer Nanopore chemistries are not expected to require an iterative correction strategy, and improved instrument throughput will enable fully assembled yeast chromosomes (Istace et al. 2016).

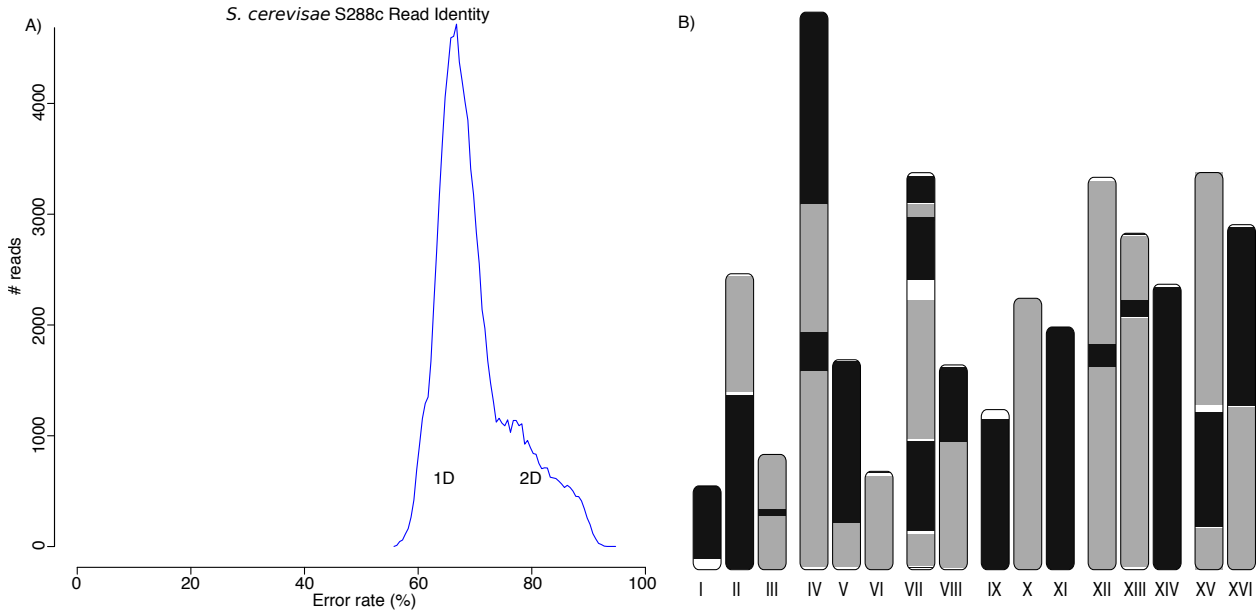

Figure 6: A highly continuous *S. cerevisae* assembly from noisy 1D and 2D MinION reads. A) A histogram of read error rates (1D and 2D) versus the reference. Alignment identity was computed only for reads with an alignment over 90% of their length. The majority of reads were below 75% identity with an overall average of 70%. B) Assembled Canu contigs were aligned to the reference and all alignments over 1 kbp in length and >90% identity were then plotted. Alternating shades indicate adjacent alignments, so each transition from gray to black represents a contig boundary or alignment breakpoint. White regions indicate regions missing from the assembly. The majority of chromosomes are in less than 3 contigs, indicating structural agreement with the reference.

Discussion

Canu is able to generate highly continuous assemblies from both PacBio and Nanopore sequencing, but signal-level polishing is required to maximize the final consensus accuracy. Such algorithms use statistical models of the sequencing process to predict base calls directly from the raw instrument data, which is a richer source of information than FASTQ Phred quality values. Currently, a PacBio base accuracy of 99.999% (QV50) is achievable with Quiver polishing (Chin et al. 2013; Koren et al. 2013), but Nanopore is limited to at most 99.9% (QV30) with Nanopolish (Loman et al. 2015) due to systematic sequencing errors (Goodwin et al. 2015). Both tools are technology specific and must be trained on each new chemistry, so future improvements are possible. Alternatively, complementary short-read sequencing can be used for consensus polishing with Pilon. On recent Nanopore sequencing data, Illumina-polished Canu assemblies can reach QV50 and exceed the base accuracy of hybrid SPAdes assemblies. Thus, the combination of Nanopore and Illumina sequencing provides a new alternative for the generation of finished microbial genomes.

Canu assembly followed by either single-molecule or short-read polishing is an efficient method for generating high-quality assemblies. Our results indicate that while Miniasm (Li 2016) can rapidly produce continuous and structurally accurate assemblies, the multiple rounds of polishing needed to produce an accurate consensus sequence becomes a computational bottleneck. Additionally, Canu is the only tool capable of assembling low-accuracy 1D Nanopore data, while scaling to gigabase-sized genomes—an important application given the pending release of high-throughput Nanopore sequencers (Oxford Nanopore Technologies 2016a). Combined with Canu's adaptive k-mer weighting strategy, the assembly of repetitive heterochromatic sequence may be possible with high-coverage, long-read nanopore sequencing.

Although Canu is designed to effectively separate divergent repeats and haplotypes, further improvements are possible. Currently, only abundance is considered for k-mer weighting, which avoids the consideration of false, repetitive overlaps. However, this same scheme could be used to improve the discrimination of minor repeat and haplotype variants by preferring haplotype-specific k-mers during sketch construction. This would increase the power of Canu's statistical overlap filter, which prevents the merging of diverged repeats and haplotypes. However, although these regions are kept separate in the assembly graph, no effort is currently made to resolve more complex repeat structures or phase multiple bubbles into larger haplotype

blocks. This is an area of future development. For repeat structures, the current algorithm can resolve any repeat copy spanned by a single read, but this is sub-optimal. For example, given a two-copy repeat, an unspanned copy can be correctly resolved so long as the other copy is spanned. For haplotype reconstruction, it would be possible to apply an approach like Falcon-Unzip (Chin et al. 2016) to the Canu assembly graph to generate phased contigs based on linked variants identified within the single-molecule reads. Alternatively, secondary information from technologies like 10x Genomics (Zheng et al. 2016) or Hi-C (Selvaraj et al. 2013) could be used to guide walks through the Canu graph. Ultimately, the integration of multiple technologies could lead to complete, *de novo* phased assemblies that span entire mammalian chromosomes from telomere to telomere.

Methods

Architecture

Canu is a modular assembly infrastructure comprised of three primary stages—correction, trimming, and assembly (Figure 1)—that can be run on a single computer or multi-node compute cluster. For multi-node runs, recommended for large genomes, Canu supports Sun Grid Engine (SGE), Simple Linux Utility for Resource Management (SLURM), Load Sharing Facility (LSF), and Portable Batch System (PBS)/Torque job schedulers. Users without access to an institutional compute cluster can run large Canu assemblies via a cloud-computing provider using toolkits such as StarCluster (http://star.mit.edu/cluster/).

As a Canu job progresses, summary statistics are updated in a set of plaintext and HTML reports. The primary data interchange between stages is FASTA or FASTQ inputs, but for efficiency each stage reads input into an indexed database, after which the original input is no longer needed. Each of the three stages begins by identifying overlaps between all pairs of input reads. Although the overlapping strategy varies for each stage, each counts k-mers in the reads, finds overlaps between the reads, and creates an indexed store of those overlaps. By default the correction stage uses MHAP (Berlin et al. 2015) and the remaining stages use overlapInCore (Myers et al. 2000). From the input reads, the correction stage generates corrected reads; the trimming stage trims unsupported bases and detects hairpin adapters, chimeric sequences, and

other anomalies; and the assembly stage constructs an assembly graph and contigs. The individual stages can be run independently or in series.

For distributed jobs, local compute resources are polled to build a list of available hosts and their specifications. Next, based on the estimated genome size, Canu will choose an appropriate range of parameters for each algorithm (e.g. number of compute threads to use for computing overlaps). Finally, Canu will automatically choose specific parameters from each allowed range so that usage of available resources is maximized. As an example, for a mammalian sized genome, Canu will choose between 1 to 8 compute threads and 4 to 16 GB memory for each overlapping job. On a grid with ten hosts, each with 18 cores and 32 GB of memory, Canu will maximize usage of all 180 cores by selecting 6 threads and 10 GB of memory per job. This process is repeated for each step, and allows automated deployment across varied cluster and host configurations, simplifying usage and maximizing resource utilization.

MinHash Overlapping

Canu uses an updated version of the MinHash Alignment Process (MHAP) for computing all-versus-all overlaps from noisy, single-molecule sequences (Berlin et al. 2015). MHAP has been further optimized for both speed and accuracy since the initial version. As described below, the most substantial algorithmic changes involve the sketching and filtering strategies. MHAP uses a two-stage overlap filter, where the first stage identifies read pairs that are likely to share an overlap and the second stage estimates the extent and quality of the overlap. For the first stage, MHAP now implements *tf-idf* weighting to prefer informative, non-repetitive k-mers. This increases sensitivity to true overlaps, while reducing the number of false, repetitive overlaps considered. For the second stage, MHAP now implements a "bottom sketch" strategy similar to Mash (Ondov et al. 2016), which significantly decreases memory usage and runtime. The Mash distance formula is also used to estimate the error rate (quality) of the identified overlaps directly from the sketches, without the need for a gapped alignment (Ondov et al. 2016). Engineering improvements include a switch to the FastUtil (fastutil 2016) hash table implementation, which resulted in a 3-fold speedup, and an increase in the maximum k-mer size from 16 to 128 to support greater specificity on low-error datasets. Overall, the new MHAP version is 10-fold faster, on average, and over 40-fold faster on mammalian genomes that the original version, while maintaining similar accuracy.

There have been several *tf-idf* formulations proposed for document and image retrieval (Manning et al. 2008), but for our purposes we use:

$$w_q = tf_q idf_q \tag{1}$$

For each read, tf_q is the number of occurrences of k-mer q in the read, and idf_q is the inverse document frequency function for q, which logarithmically scales the inverse overall frequency of q observed across all reads. Specifically, for all k-mers in the input read set, let f_{max} be the maximum observed frequency, f_{min} be the minimum observed frequency, and f_q be the frequency of a specific k-mer q. By default, only 0.000005 of the most abundant k-mers are recorded, and all others are assigned f_{min}. We define idf_q as:

$$idf_q = T\left(\log(\frac{f_{max}}{f_q} - a)\right) \tag{2}$$

The parameter $a \in [0,1]$ controls how strongly less common k-mers are preferred in relation to the more common ones, and T linearly transforms the values between 1 and w_{max}, the maximum allowed weight. For a general positive floating point number, (Chum et al. 2008) provided a formula for directly computing the w-weighted hash value for MinHash ranking. However, this formula requires computing $s \cdot L$ logarithms to generate a sketch, which is computationally expensive (where s is the sketch size and L is the read length). Instead, we discretize the *tf-idf* to a limited range using rounding, which requires at most $s \cdot L \cdot w_{max}$ random number computations, which is comparatively faster. We use $w_{max} = 3$ and $a = 0.9$ by default as a compromise between speed and performance.

Recall that MinHash selects which k-mers will be included in the sketch on the basis of their hash value. In the original MHAP implementation, a set Γ of s hash functions is defined for a sketch S of size s. Each sketch entry S_i is defined as the minimum-valued k-mer after applying the hash function Γ_i to all k-mers in the read. The resulting set of s minimum-valued k-mers, or min-mers, comprise the sketch. Given a discrete *tf-idf* weight w_q for each k-mer, we now modify the MinHash computation by applying w_q hash functions $\{\Gamma_{i,1}, \ldots, \Gamma_{i,w_q}\}$ per entry, rather than the single Γ_i as before. For each sketch entry S_i, the min-mer is then chosen as the minimum hash value computed across all functions. Because highly weighted k-mers are hashed more times, this increases the chance that they will be chosen as a min-mer. In order to properly match the

same k-mers with different weights, we index k-mers using their fixed MurmurHash3 hash values (Appleby 2014), and the weighted values are only used to determine inclusion in the read sketches. The *tf-idf* approach replaces the previous approach, based on traditional all-or-nothing filtering of repetitive k-mers. We evaluated multiple scoring approaches including *tf-idf*, *idf* only (down-weighting common words), and no weighting on several bacterial and eukaryotic genomes. Both *tf-idf* and *idf* outperformed unweighted comparisons in terms of the resulting assembly continuity and accuracy, and were comparable to each other. We therefore utilize *tf-idf* by default due to its common use in the natural language field and other MinHash applications (Chum et al. 2008).

The updated MHAP version also implements bottom sketching for the second-stage filter (Ondov et al. 2016). In contrast to the first-stage filter, which uses multiple hash functions (Broder et al. 2000), bottom sketching uses a single hash function, from which the s minimum values are retained as the sketch (Broder 1997). The former approach has the advantage that the Jaccard similarity can be computed for 1 versus N reads by a series of s hash table lookups. In bottom sketching, each comparison requires an $O(s)$ merge operation, but as a benefit, any substring of the original string can be sketched by simply eliminating the min-mers from the original sketch that do not occur in the substring. For the bottom sketch, we now store a constant number of k-mers per read (default 1,500), and directly estimate the overlap error rate from these sketches using the Mash distance. The overlapping region is estimated as previously (Berlin et al. 2015), but also using the bottom sketch k-mers.

Parallel Overlap Sort and Index

The downstream algorithms require efficient access to all overlaps for a single read, so the overlaps are organized using an indexed on-disk structure where all overlaps for a single read are listed sequentially. Canu parallelizes overlap computation into multiple jobs, each generating a compressed file of binary encoded overlaps and a file recording the number of overlaps for each read in that file. These files are combined into the master structure using a parallel bucket sort (Supplementary Figure S24). Since each read will have a different number of overlaps, and all overlaps for a given read must be in the same bucket in order for the bucket to be sorted, the number of overlaps per read is used to compute the ranges of reads assigned to each bucket. The size of a bucket is chosen such that each contains the same number of overlaps, and no bucket is

larger than some specified maximum size. In parallel, each file of compressed overlaps is rewritten to a set of uniquely named buckets, and overlaps are duplicated and added to the appropriate bucket (e.g. read A overlaps B *and* read B overlaps A). Note that as each input file creates its own set of buckets, no synchronization is needed between jobs. When all overlaps are copied into buckets, each bucket is loaded into memory, sorted, and output to a uniquely named file. Each bucket holds all of (and only) the overlaps for the range of assigned reads. Finally, an index describing the file and offset location for each read is created.

Read Correction

Canu uses all-versus-all overlap information to correct individual reads. However, simply computing a consensus representation for each read using all overlaps could result in masking copy-specific repeat variants. Therefore, Canu uses two filtering steps to determine which overlaps should be selected to correct each individual read. The first is a global filter where each read chooses where it will supply correction evidence, and the second is a local filter where each read accepts or rejects the evidence supplied by other reads. This strategy attempts to overcome biases due to sequence quality and repeats. For example, reads with higher than average sequencing quality would tend to dominate the correction, regardless of if they were from the correct repeat copy. To prevent this, each read is only allowed to contribute to the correction of C other reads, where C is the expected read depth. The global filter scores each overlap (*overlap_length* * *identity*), and keeps only the C best overlaps for each read. This same concept was used in (Koren et al. 2012). From this list, the local filter then selects the $2C$ best overlaps to each read for use in correction. The second filter is primarily a computational optimization.

Before computing the corrected sequence, the all-pair overlaps are used to predict the expected length of each read after correction (i.e. accounting for reads with partial or no overlaps). From these estimates, the longest reads up to a user-specified coverage depth are processed for correction. Corrected reads are generated using a modified implementation of the "falcon_sense" algorithm (Chin et al. 2016), which parallelizes the pairwise alignment step and removes and maximum read length limits. For a given read to be corrected, overlapping reads are aligned to it using Myers' ND algorithm (Myers 1986). A directed acyclic graph (DAG) is created from the alignments, and the highest weight path is followed to generate a corrected

sequence (Chin et al. 2016). Edges with weight less than four are omitted, which will split the original read when there is insufficient evidence for correction.

Overlap Based Trimming

After correction, reads are trimmed by re-computing overlaps for the corrected reads and removing sequence that is not supported by other reads. The prior correction stage also trims low-coverage regions, but these initial overlaps are constructed without constructing a gapped alignment, which can result in imprecise trim points. When overlapping the corrected reads for trimming, a gapped alignment is computed for each overlap, and the trim points can be identified more precisely. Overlap-based trimming (OBT) was first described by Miller *et al.* (Miller et al. 2008) and Prüfer *et al.* (Prüfer et al. 2012), which focused on trimming Sanger, 454 and Illumina reads. Long reads with uniform error allow the algorithm to be simplified. Each read is trimmed to the largest portion covered to at least depth C by overlaps of at most E error and minimum length L. The parameters are technology specific and set to empirically derived defaults.

Once reads are trimmed, a second pass is made to detect any technology specific flaws, e.g. undetected hairpin adapters and chimeras (Eid et al. 2009; Jain et al. 2015). A hairpin adapter is detected by identifying when multiple reads have both forward and reverse overlaps around a common (short) sequence and there are few reads spanning this region. A chimeric junction is similarly detected by identifying a region with few, if any, spanning reads. In both cases, the original read is trimmed to the largest supported region.

Overlap Error Adjustment

After trimming and before graph construction, Canu recomputes overlaps and makes a final attempt at detecting sequencing errors. This algorithm was first used in Holt *et al.* (Holt et al. 2002). The intuition is to improve separation between true sequencing differences (e.g. diverged repeats or haplotype) and false differences due to random sequencing error. Each read is corrected by a majority vote of its overlapping alignments, preserving differing bases only if there is sufficient support from other reads for this variation. The read sequence itself is not changed (doing so would invalidate the computed overlaps), but the reported error rate for each overlap is adjusted based on the alignment that would be generated had the sequencing errors been resolved. The algorithm requires two passes through the overlaps, the first pass detects

probable sequencing errors in reads and the second pass applies those changes temporarily to reads to re-compute alignments and update the computed error rates.

Graph Construction

The Bogart module builds an assembly graph using a variant of the "best overlap graph" strategy from (Miller et al. 2008). Overlaps are described as *containment*, if all bases in one read are aligned to another read, or *dovetail*, if involving only the ends of both reads. By definition, at least two read ends must be present in the alignment. A "best" overlap is the longest dovetail overlap to a given read end. Each read has two best overlaps, one on the 5' end and one on the 3' end. In the original method, best overlaps were picked from all overlaps up to a user supplied overlap error rate cutoff. In Bogart, best overlaps are picked after several filtering steps remove abnormally high-error overlaps, potential chimeric reads, and reads whose overlaps indicate a possible sequence anomaly. This results in a cleaner and more accurate graph construction.

After correction, trimming, and overlap error adjustment, all computed overlaps are used to pick an initial set of best edges. This set of best edges is used to compute the median and median absolute deviation (MAD) of the overlap error rate. This distribution represents the residual read error left after all prior corrections, and a low average overlap error rate cutoff indicates good sequencing data and successful correction. A maximum overlap error rate cutoff is automatically computed from this distribution as six MADs away from the median, and overlaps with an error greater than this cutoff are not used during graph construction. This cutoff, which is typically less than 2% for good PacBio data, determines the ability of the algorithm to separate closely related repeats and haplotypes.

In addition to filtering poor overlaps, Bogart filters suspicious reads that may have evaded proper trimming and correction. First, reads that are not fully covered by overlaps below the overlap error rate cutoff are flagged as potentially chimeric and excluded from graph construction. Second, best overlaps are usually mutual, i.e. the best overlap from *A* is to *B* and the best overlap from *B* is to *A*. For a pair of reads, non-mutual best overlaps are often caused by Indels, making the overlap length slightly longer or shorter compared to the mutual best overlap. Thus, reads with a large overlap size difference are also excluded (Supplementary Figure S25).

The resulting set of reads and best overlaps define the best overlap graph. Initial contigs are then constructed from the best overlap graph as in (Miller et al. 2008), and an error rate

profile is generated for each contig from the average error rate of overlaps used to build it. This error profile is recomputed after each phase of the algorithm, and is used to determine if external reads have valid overlaps to the contig.

Bogart next attempts to include contained (Fasulo et al. 2002) and previously filtered reads into the contigs. All overlaps to these read are used to compute a set of potential contig placements, scored by the average overlap error rate. If this average error rate exceeds the pre-computed error profile for the contig region the read is likely from a diverged repeat or a heterozygous variant, and the placement is rejected. The placement with the lowest average error is accepted, and the read is placed. This strategy differs from the original strategy from (Miller et al. 2008) that placed contained reads based on the highest quality containment overlap, which could incorrectly place a read when the true location had no container read. Reads that remain unplaced after this phase are output as "unassembled."

An assembly bubble occurs when there is more than one reconstruction of a specific locus caused by haplotype differences (Fasulo et al. 2002; Zerbino and Birney 2008; Koren et al. 2011; Nijkamp et al. 2013; Chin et al. 2016). Small differences, tens of base pairs in size, are typically not detectable from overlaps alone because the difference is insignificant compared to the size of the overlap. Larger differences can result in two, mostly redundant, contigs covering the same locus. The more common haplotype is often reconstructed in a large contig spanning the locus, and the less common haplotype as just the variant region (the bubble). Currently, contigs with fewer than a minimum threshold of reads, or with more than 75% of the reads with an overlap to some other contig, are considered potential bubbles. Reads in these contigs are then placed, using the mechanism for placing unplaced reads as above, into all other contigs where possible using heuristics. Improved mechanisms for resolving bubbles within the assembly graph, and ultimately producing a fully phased assembly, is an area of ongoing research and left for future work.

Despite careful filtering, the greedy construction algorithm remains prone to error and the graph will be missing edges compared to a full string graph representation, so a final step is required to add missing edges and break incorrectly assembled contigs. Using the all-pairs overlap information, every assembled contig is annotated with compatible read placements, again using the read placement mechanism and all reads from non-bubble contigs. Only overlaps that meet the global and local contig error rate thresholds are considered. The resulting annotated

regions indicate alternative branch points in the full overlap graph, and a correct contig reconstruction is confirmed by the presence of spanning reads or overlaps. Unresolved regions are marked as repeats, the contig is split, and additional edges are added to form the final assembly graph.

Contig Consensus

Canu generates a consensus sequence for each contig using a modified version of the "pbdagcon" algorithm (Chin et al. 2013). Briefly, a template sequence is constructed for each contig by splicing reads together from approximate positions based on the best overlap path. This template is accurate within individual reads, as they have previously been error-corrected, but may have Indel errors at read boundaries due to inaccuracy in the overlap positions. To correct this, all reads in the contig are aligned to the template sequence in parallel using Myers' ND algorithm (Myers 1986) and added to a DAG. The DAG is then used to call a consensus sequence as in (Chin et al. 2013).

Data Access

The *Bacillus anthracis* Sterne sequencing data can be accesed through BioProject PRJXXXX and the *Yersinia pestis* CO92 sequencing data can be accessed through PRJXXXX. All other sequencing is publically available and listed in Supplementary Note 2.

Acknowledgements

We thank Celera and Pacific Biosciences for open source software that was critical for the development of Canu, and also John Urban and all other Canu users who provided early testing and feedback on the software. This research was supported in part by the Intramural Research Program of the National Human Genome Research Institute, National Institutes of Health, and utilized the computational resources of the NIH HPC Biowulf cluster (http://hpc.nih.gov).

References

Antipov D, Korobeynikov A, McLean JS, Pevzner PA. 2016. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics* **32**: 1009-1015.

Appleby A MurmurHash3 http://code.google.com/p/smhasher/wiki/MurmurHash3 (2014).

Attrill H, Falls K, Goodman JL, Millburn GH, Antonazzo G, Rey AJ, Marygold SJ, FlyBase C. 2016. FlyBase: establishing a Gene Group resource for Drosophila melanogaster. *Nucleic Acids Res* **44**: D786-792.

Berlin K, Koren S, Chin C-S, Drake JP, Landolin JM, Phillippy AM. 2015. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat Biotech* **33**: 623-630.

Bresler G, Bresler M, Tse D. 2013. Optimal assembly for high throughput shotgun sequencing. *BMC Bioinformatics* **14 Suppl 5**: S18.

Broder AZ. 1997. On the resemblance and containment of documents. *Compression and Complexity of Sequences 1997 Proceedings*: 21-29.

Broder AZ, Charikar M, Frieze AM, Mitzenmacher M. 2000. Min-wise independent permutations. *Journal of Computer and System Sciences* **60**: 630-659.

Burton JN, Adey A, Patwardhan RP, Qiu R, Kitzman JO, Shendure J. 2013. Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nature biotechnology* **31**: 1119-1125.

Chakraborty M, Baldwin-Brown JG, Long AD, Emerson JJ. 2016. Contiguous and accurate de novo assembly of metazoan genomes with modest long read coverage. *biorxiv*.

Chin CS, Alexander DH, Marks P, Klammer AA, Drake J, Heiner C, Clum A, Copeland A, Huddleston J, Eichler EE et al. 2013. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature methods* **10**: 563-569.

Chin CS, Peluso P, Sedlazeck FJ, Nattestad M, Concepcion GT, Clum A, Dunn C, O'Malley R, Figueroa-Balderas R, Morales-Cruz A et al. 2016. Phased Diploid Genome Assembly with Single Molecule Real-Time Sequencing. *biorxiv*.

Chum O, Philbin J, Zisserman A. 2008. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. *BMVC* **810**: 812-815.

Eid J, Fehr A, Gray J, Luong K, Lyle J, Otto G, Peluso P, Rank D, Baybayan P, Bettman B et al. 2009. Real-time DNA sequencing from single polymerase molecules. *Science* **323**: 133-138.

Ewing B, Green P. 1998. Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Res* **8**: 186-194.

fastutil: Fast & compact type-specific collections for Java™. http://fastutil.di.unimi.it (2016).

Fasulo D, Halpern A, Dew I, Mobarry C. 2002. Efficiently detecting polymorphisms during the fragment assembly process. *Bioinformatics* **18 Suppl 1**: S294-302.

Felsenfeld A, Peterson J, Schloss J, Guyer M. 1999. Assessing the quality of the DNA sequence from the Human Genome Project. *Genome Res* **9**: 1-4.

Goodwin S, Gurtowski J, Ethe-Sayers S, Deshpande P, Schatz MC, McCombie WR. 2015. Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Res* **25**: 1750-1756.

Gordon D, Huddleston J, Chaisson MJ, Hill CM, Kronenberg ZN, Munson KM, Malig M, Raja A, Fiddes I, Hillier LW et al. 2016. Long-read sequence assembly of the gorilla genome. *Science* **352**: aae0344.

Hackl T, Hedrich R, Schultz J, Forster F. 2014. proovread: large-scale high-accuracy PacBio correction through iterative short read consensus. *Bioinformatics* **30**: 3004-3011.

Hastie AR, Dong L, Smith A, Finklestein J, Lam ET, Huo N, Cao H, Kwok PY, Deal KR, Dvorak J et al. 2013. Rapid genome mapping in nanochannel arrays for highly complete

and accurate de novo sequence assembly of the complex Aegilops tauschii genome. *PLoS One* **8**: e55864.

Holt RA Subramanian GM Halpern A Sutton GG Charlab R Nusskern DR Wincker P Clark AG Ribeiro JM Wides R et al. 2002. The genome sequence of the malaria mosquito Anopheles gambiae. *Science* **298**: 129-149.

Hoskins RA, Carlson JW, Wan KH, Park S, Mendez I, Galle SE, Booth BW, Pfeiffer BD, George RA, Svirskas R et al. 2015. The Release 6 reference sequence of the Drosophila melanogaster genome. *Genome Res* **25**: 445-458.

International Human Genome Sequencing C. 2004. Finishing the euchromatic sequence of the human genome. *Nature* **431**: 931-945.

Istace B, Friedrich A, d'Agata L, Faye S, Payen E, Beluche O, Caradec C, Davidas S, Cruaud C, Liti G et al. 2016. de novo assembly and population genomic survey of natural yeast isolates with the Oxford Nanopore MinION sequencer. *biorxiv*.

Istrail S, Sutton GG, Florea L, Halpern AL, Mobarry CM, Lippert R, Walenz B, Shatkay H, Dew I, Miller JR et al. 2004. Whole-genome shotgun assembly and comparison of human genome assemblies. *Proc Natl Acad Sci U S A* **101**: 1916-1921.

Jain M, Fiddes IT, Miga KH, Olsen HE, Paten B, Akeson M. 2015. Improved data analysis for the MinION nanopore sequencer. *Nature methods* **12**: 351-356.

Judge K, Hunt M, Reuter S, Tracey A, Quail MA, Parkhill J, Peacock SJ. 2016. Comparison of bacterial genome assembly software for MinION data. *biorxiv*.

Kaplan N, Dekker J. 2013. High-throughput genome scaffolding from in vivo DNA interaction frequency. *Nature biotechnology* **31**: 1143-1147.

Kim KE, Peluso P, Babayan P, Yeadon PJ, Yu C, Fisher WW, Chin C-S, Rapicavoli NA, Rank DR, Li J et al. 2014. Long-read, whole-genome shotgun sequence data for five model organisms. *Scientific Data* **1**.

Koren S, Harhay GP, Smith TP, Bono JL, Harhay DM, McVey SD, Radune D, Bergman NH, Phillippy AM. 2013. Reducing assembly complexity of microbial genomes with single-molecule sequencing. *Genome Biol* **14**: R101.

Koren S, Phillippy AM. 2014. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Curr Opin Microbiol* **23C**: 110-120.

Koren S, Schatz MC, Walenz BP, Martin J, Howard JT, Ganapathy G, Wang Z, Rasko DA, McCombie WR, Jarvis ED et al. 2012. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology* **30**: 693-700.

Koren S, Treangen TJ, Pop M. 2011. Bambus 2: Scaffolding Metagenomes. *Bioinformatics* **27**: 2964-2971.

Kurtz S, Phillippy A, Delcher AL, Smoot M, Shumway M, Antonescu C, Salzberg SL. 2004. Versatile and open software for comparing large genomes. *Genome biology* **5**: R12-R12.

Lee H, Gurtowski J, Yoo S, Marcus S, McCombie WR, Schatz M. 2014. Error correction and assembly complexity of single molecule sequencing reads. *biorxiv* doi:10.1101/006395.

Li H. 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32**: 2103-2110.

Loman NJ, Quick J, Simpson JT. 2015. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature methods* **12**: 733-735.

Manning CD, Raghavan P, Schütze H. 2008. Scoring, term weighting and the vector space model. *Introduction to Information Retrieval* **100**: 2-4.

Metz CW. 1914. Chromosome studies in the Diptera. *Journal of Experimental Zoology* **17**: 45-59.

Miller JR, Delcher AL, Koren S, Venter E, Walenz BP, Brownley A, Johnson J, Li K, Mobarry C, Sutton G. 2008. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics* **24**: 2818-2824.

Myers EW. 1986. An O(ND) difference algorithm and its variations. *Algorithmica* **1**: 251-266.

Myers EW. 2005. The fragment assembly string graph. *Bioinformatics (Oxford, England)* **21 Suppl 2**: ii79-85.

Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, Flanigan MJ, Kravitz SA, Mobarry CM, Reinert KH, Remington KA et al. 2000. A whole-genome assembly of Drosophila. *Science* **287**: 2196-2204.

Myers G. 2014. Efficient Local Alignment Discovery amongst Noisy Long Reads. *Algorithms in Bioinformatics* **8701**: 52-67.

Nagarajan N, Pop M. 2009. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J Comput Biol* **16**: 897-908.

Nijkamp JF, Pop M, Reinders MJT, de Ridder D. 2013. Exploring variation-aware contig graphs for (comparative) metagenomics using MaryGold. *Bioinformatics* **29**: 2826-2834.

Ning Z, Cox AJ, Mullikin JC. 2001. SSAHA: a fast search method for large DNA databases. *Genome Res* **11**: 1725-1729.

Ondov BD, Treangen TJ, Melsted P, Mallonee AB, Bergman NH, Koren S, Phillippy AM. 2016. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biol* **17**: 132.

Ono Y, Asai K, Hamada M. 2013. PBSIM: PacBio reads simulator--toward accurate genome assembly. *Bioinformatics* **29**: 119-121.

Oxford Nanopore Technologies PromethION: small benchtop system for high throughput real-time biological analyses and allowing large sample numbers. https://www.nanoporetech.com/products-services/promethion (2016a).

Oxford Nanopore Technologies Rapid Sequencing. https://cws.nanoporetech.com/rapidsequencing (2016b).

Pacific Biosciences DevNet Datasets https://github.com/PacificBiosciences/DevNet/wiki/Datasets (2014).

Pacific Biosciences Quiver Frequently Asked Questions. https://github.com/PacificBiosciences/GenomicConsensus/blob/master/doc/QuiverFAQ.rst (2015).

Phillippy AM, Schatz MC, Pop M. 2008. Genome assembly forensics: finding the elusive mis-assembly. *Genome biology* **9**: R55-R55.

Prüfer K, Munch K, Hellmann I, Akagi K, Miller JR, Walenz B, Koren S, Sutton G, Kodira C, Winer R. 2012. The bonobo genome compared with the chimpanzee and human genomes. *Nature*.

Quick J, Loman NJ First SQK MAP 006 experiment. http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/ (2015).

Quick J, Loman NJ Nanopore R9 rapid run data release. http://lab.loman.net/2016/07/30/nanopore-r9-data-release/ (2016).

Ross MG, Russ C, Costello M, Hollinger A, Lennon NJ, Hegarty R, Nusbaum C, Jaffe DB. 2013. Characterizing and measuring bias in sequence data. *Genome Biol* **14**: R51.

Salmela L, Rivals E. 2014. LoRDEC: accurate and efficient long read error correction. *Bioinformatics* **30**: 3506-3514.

Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, Treangen TJ, Schatz MC, Delcher AL, Roberts M. 2012. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research* **22**: 557-567.

Schmutz J, Wheeler J, Grimwood J, Dickson M, Yang J, Caoile C, Bajorek E, Black S, Chan YM, Denys M et al. 2004. Quality assessment of the human genome sequence. *Nature* **429**: 365-368.

Schneider GF, Dekker C. 2012. DNA sequencing with nanopores. *Nature biotechnology* **30**: 326-328.

Selvaraj S, J RD, Bansal V, Ren B. 2013. Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing. *Nature biotechnology* **31**: 1111-1118.

Sovic I, Krizanovic K, Skala K, Sikic M. 2016. Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads. *Bioinformatics* doi:10.1093/bioinformatics/btw237.

Stevens NM. 1912. The chromosomes in Drosophila ampelophila. *Proceedings of the 7th International Zoological Congress, Boston*: 380-381.

Sutton GG, White O, Adams MD, Kerlavage AR. 1995. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology* **1**: 9-19.

Tørresen OK, Star B, Jentoft S, Reinar WB, Grove H, Miller JR, Walenz BP, Knight J, Ekholm JM, Peluso P et al. 2016. An improved genome assembly uncovers a prolific tandem repeat structure in Atlantic cod. *biorxiv*.

Travers KJ, Chin CS, Rank DR, Eid JS, Turner SW. 2010. A flexible and efficient template format for circular consensus sequencing and SNP detection. *Nucleic Acids Res* **38**: e159.

Ukkonen E. 1992. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science* **92**: 191-211.

Walker BJ, Abeel T, Shea T, Priest M, Abouelliel A, Sakthikumar S, Cuomo CA, Zeng Q, Wortman J, Young SK et al. 2014. Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS One* **9**: e112963.

Wang H, Avican K, Fahlgren A, Erttmann SF, Nuss AM, Dersch P, Fallman M, Edgren T, Wolf-Watz H. 2016. Increased plasmid copy number is essential for Yersinia T3SS function and virulence. *Science* **353**: 492-495.

Wick RR, Schultz MB, Zobel J, Holt KE. 2015. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* **31**: 3350-3352.

Ye C, Hill C, Ruan J, (Sam)Ma Z. 2014. DBG2OLC: Efficient assembly of large genomes using the compressed overlap graph. *arXiv preprint arXiv:14102801*.

Zerbino DR, Birney E. 2008. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research* **18**: 821-829.

Zheng GX, Lau BT, Schnall-Levin M, Jarosz M, Bell JM, Hindson CM, Kyriazopoulou-Panagiotopoulou S, Masquelier DA, Merrill L, Terry JM et al. 2016. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. *Nature biotechnology* **34**: 303-311.