ARTICLES
https://doi.org/10.1038/s41587-019-0072-8
nature biotechnology

Assembly of long, error-prone reads using repeat graphs

Mikhail Kolmogorov[1], Jeffrey Yuan[2], Yu Lin[3] and Pavel A. Pevzner[1*]

Accurate genome assembly is hampered by repetitive regions. Although long single molecule sequencing reads are better able to resolve genomic repeats than short-read data, most long-read assembly algorithms do not provide the repeat characterization necessary for producing optimal assemblies. Here, we present Flye, a long-read assembly algorithm that generates arbitrary paths in an unknown repeat graph, called disjointigs, and constructs an accurate repeat graph from these error-riddled disjointigs. We benchmark Flye against five state-of-the-art assemblers and show that it generates better or comparable assemblies, while being an order of magnitude faster. Flye nearly doubled the contiguity of the human genome assembly (as measured by the NGA50 assembly quality metric) compared with existing assemblers.

Genome assembly is the process of reconstructing genomes from DNA sequence reads. In repetitive regions of the genome, accurately assembling short reads is challenging and can lead to inaccurate or unresolved assemblies. Single molecule sequencing (SMS) long-read technologies (such as Pacific Biosciences and Oxford Nanopore) have been used to improve the resolution of repetitive genomic regions, but many long stretches of repetitive DNA remain intractable to these approaches. Current SMS assemblers, such as PBcR[1–3], Falcon[4], Miniasm[5], ABruijn[6], HINGE[7], Canu[8], and Marvel[9], have been used to successfully resolve some repeat regions across complex genomes, but correct assembly of long reads in long and highly repetitive genomic regions remains challenging. As a result, long-read technologies are often complemented by proximity ligation techniques (Hi-C)[10] and optical[11] mapping data to improve the contiguity of assemblies.

The de Bruijn graph has been used by short-read assembly approaches to represent genomic repeats as a repeat graph. Previous studies have demonstrated the value of this approach for improving the accuracy of genome assembly[12]. Recently, long-read assemblers such as ABruijn[6] and HINGE[7], which capitalize on a similar de Bruijn graph-based approach, have been developed. Most short-read assemblers construct the de Bruijn graph based on all *k*-mers in reads and further transform it into a simpler de Bruijn assembly graph[13]. This approach collapses multiple instances of the same repeat into a single path in the assembly graph and represents the genome as a genome tour, which visits each edge in the assembly graph. However, in the case of SMS reads, the key assumption of the de Bruijn graph approach—that most *k*-mers from the genome are preserved in multiple reads—does not hold. As a result, various challenges that have been addressed for short-read assembly, such as how to deal with the fragmented de Bruijn graph and how to transform it into an assembly graph, remain largely unaddressed in long-read assemblers.

Here we describe the Flye algorithm for accurately assembling long reads (Fig. 1). Unlike existing assemblers that attempt to generate contigs, Flye initially generates disjointigs that represent concatenations of multiple disjoint genomic segments, concatenates all error-prone disjointigs into a single string (in an arbitrary order), constructs an accurate assembly graph from the resulting concatenate, uses reads to untangle this graph, and resolves bridged repeats (which are bridged by some reads in the repeat graph). Afterwards, it uses the repeat graph to resolve unbridged repeats (which are not bridged by any reads) using small differences between repeat copies and then outputs accurate contigs formed by paths in this graph.

We benchmark Flye against five state-of-the-art SMS assemblers (Falcon, Miniasm, HINGE, Canu, and MaSuRCA) and show that it generates more accurate and contiguous assemblies and provides valuable information to aid in assembly finishing. Flye also reconstructs the mosaic structure of segmental duplications (SDs)—a difficult problem even for finished genomes[14,15].

Results

Repeat graph construction. Repeats in a genome are often represented as pair-wise local alignments and visualized as alignment-paths in a two-dimensional dot-plot of a genome. This pair-wise representation is limited since it does not contribute to solving the repeat characterization problem[12,16]. In contrast, the repeat graph compactly represents all repeats in a genome and reveals their mosaic structure[12,14]. Assembly graph construction represents a special case of the repeat graph construction problem.

Figure 2 outlines the algorithm for constructing the repeat graph of a finished (complete) genome. Flye applies this algorithm to construct the repeat graph of a pseudo-genome formed by concatenating all disjointigs (formed at the previous stage of the pipeline) in an arbitrary order. The Methods section explains why the resulting graph provides the correct representation of the assembled genome (as if it had been constructed from a complete genome) and describes additional algorithmic details.

Resolving unbridged repeats with Flye. Flye utilizes the constructed repeat graph for the resolution of unbridged repeats. Resolving unbridged and nearly identical repeats using SMS reads is a difficult problem since error-prone SMS reads make it difficult to distinguish repeat copies with divergence below 10%. As a result, SMS assemblers often fail to resolve unbridged repeats, which are common even in bacterial genomes[7,17]. This challenge is related to

[1]Department of Computer Science and Engineering, University of California, San Diego, CA, USA. [2]Graduate Program in Bioinformatics and Systems Biology, University of California, San Diego, CA, USA. [3]Research School of Computer Science, Australian National University, Canberra, Australian Capital Territory, Australia. *e-mail: ppevzner@ucsd.edu

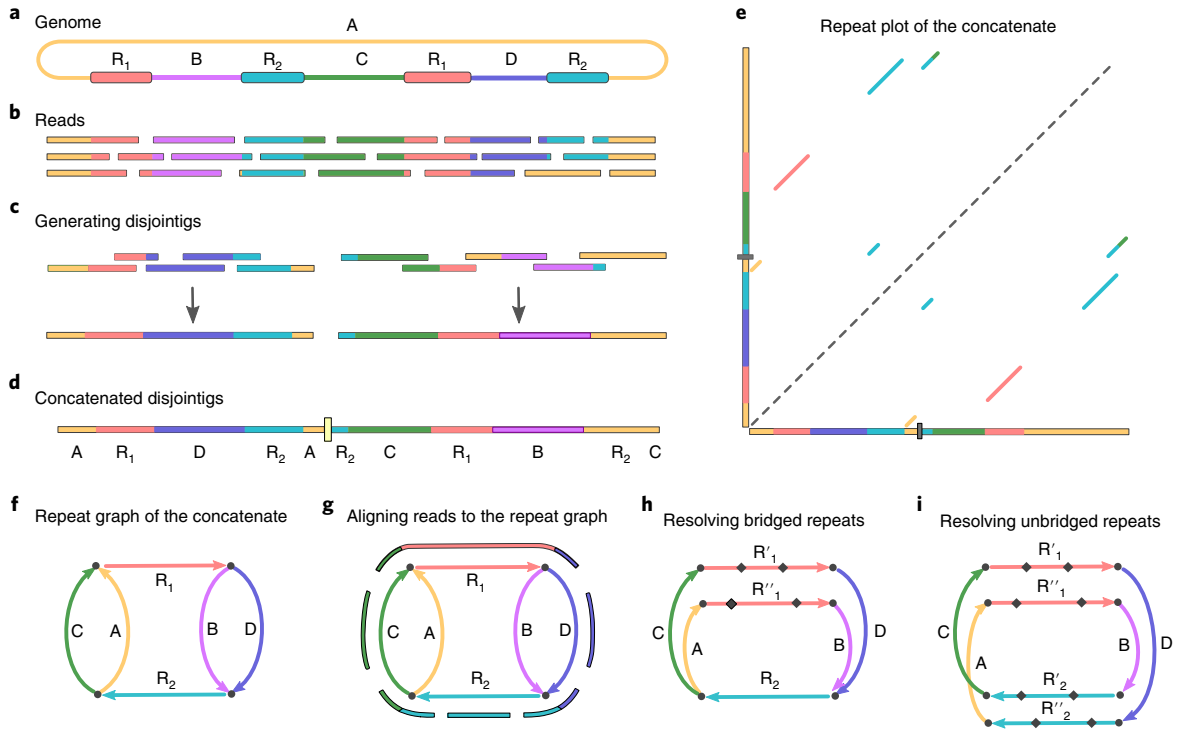

Fig. 1 | Flye outline. a, A 'genome' with two 99% identical copies of a repeat R_1 and two 99% identical copies of a repeat R_2. Segments A, B, C, and D represent non-repetitive regions. **b**, A set of reads sampled from the genome. **c**, Two (misassembled) disjointigs AR_1DR_2A and $R_2CR_1BR_2C$ derived from the reads. **d**, Concatenate of the disjointigs. **e**, Repeat plot of the concatenate. **f**, Repeat graph constructed by 'gluing' vertices in the concatenate according to the repeat plot. For each two-dimensional point (x, y) in the repeat plot, we glue vertices x and y in the concatenate. **g**, Aligning reads against the repeat graph. **h**, Resolving the bridged repeat R_1 and reconstructing its two copies R'_1 and R''_1. The differences between each copy of this repeat and the consensus of this repeat are shown as small diamonds. **i**, Resolving the unbridged repeat R_2 with two slightly diverged copies. Supplementary Note 4 describes the Flye assembly of a simulated genome modeled after the genome shown in **a**.

the challenge of constructing phased diploid genome assemblies[4] and overlap-filtering for repeat resolution[8,18]. The repeat graph constructed by Flye offers an approach for resolving unbridged repeats based on analyzing the topology of the repeat graph.

Figure 3 shows an unbridged repeat REP as an edge in the assembly graph. It would be impossible to resolve this repeat (that is, to pair each incoming edge into the initial vertex of REP with the corresponding outgoing edge from the terminal vertex of REP) if its two copies were identical. However, since there exist variations between these copies, it becomes possible to transform the single sequence REP into two different repeat instances, REP_1 and REP_2, as shown in Fig. 3. The Methods section describes how Flye resolves unbridged repeats by (1) identifying variations between repeat copies, (2) matching each read with a specific repeat copy using these variations, and (3) using these reads to derive a distinct consensus sequence for each repeat copy.

Benchmarking Flye. We benchmarked Flye against SMS assemblers Canu, Falcon, HINGE, Miniasm, and MaSuRCA using six datasets. We used QUAST[19] to evaluate all assemblers (Supplementary Note 1). Since Miniasm returns assemblies with a much larger number of mismatches and indels than other assemblers, it is not well suited for a reference-based quality evaluation with QUAST. To make a fair comparison, we ran the ABruijn contig-polishing module[6] on the Miniasm output to improve the accuracy of its contigs (referred to as Miniasm + ABruijn).

Benchmarking with the BACTERIA dataset. The dataset consists of 21 sets of Pacific Biosciences (PacBio) reads from the National Collection of Type Cultures (NCTC). These NCTC sets were studied in detail in ref. [7] and used to benchmark various assemblers. We only benchmarked Flye against HINGE on these datasets, since HINGE outperformed the other assemblers on bacterial genomes[7]. We ignored small connected components in the bacterial assembly graphs (which represent plasmids that do not share repeats with chromosomes) and classified an assembly as (1) complete if the assembly graph consists of a single loop-edge representing a circular chromosome, (2) semicomplete if the assembly graph contains multiple edges but there exists a single Chinese postman tour in this graph[20], and (3) tangled if the assembly graph is neither complete nor semicomplete.

While HINGE does not distinguish between complete and semicomplete assemblies, we argue that ignoring this separation may lead to assembly errors. Indeed, a single Chinese postman tour in a semicomplete assembly graph results in a unique assembly only in the case of unichromosomal genomes without any plasmids that share repeats with the chromosome (repeat-sharing plasmids). In the case of multichromosomal genomes or in the case of repeat-sharing plasmids, there exist multiple possible assemblies from a semicomplete assembly graph. Since ~10% of known bacterial genomes are multichromosomal and since a large fraction of unichromosomal genomes have repeat-sharing plasmids[21], the assumption that a semicomplete assembly graph results in a complete genome reconstruction may lead to errors.

Before resolving unbridged repeats, Flye assembled the genomes from the BACTERIA dataset into 4 complete, 1 semicomplete, and 16 tangled assembly graphs. After resolving unbridged repeats, the Flye assemblies resulted in 8 complete, 5 semicomplete, and 8 tangled assembly graphs with the number of edges varying from 3 to 25. Supplementary Fig. 1 shows examples of assembly graphs generated

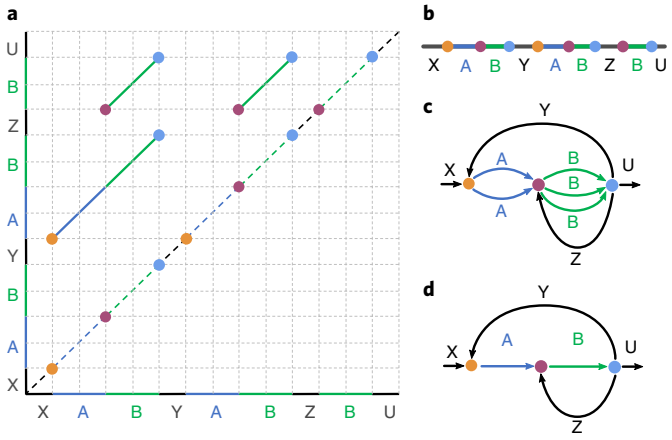

Fig. 2 | Constructing the approximate repeat graph from local self-alignments. a, Alignment-paths for all local self-alignments within a genome XABYABZBU formed by segments X, A, B, Y, Z, and U. Three instances of a mosaic repeat (AB, AB, and B) are represented as diagonal alignment-paths in the repeat plot. The self-alignment of the entire genome is shown by the main (dotted) diagonal. Alignment endpoints are clustered together if their projections on the main diagonal coincide or are close to each other (clusters of closely located endpoints for the distance threshold $d = 0$ are painted with the same color). For example, the right-most endpoints (shown in blue) of all three alignments form a single cluster because two of them have the same vertical projection and two of them have the same horizontal projection on the main diagonal. This clustering reveals three clusters (yellow, purple, and blue) with eight projections to the main diagonal. **b**, Projections of the clustered endpoints on the main diagonal define eight vertices (breakpoints) that will be used for constructing the approximate repeat graph. **c**, Breakpoints that belong to the same clusters are glued together. **d**, Gluing parallel edges in the resulting graph produces the approximate repeat graph.

by Flye and HINGE, and Supplementary Table 1 illustrates that Flye and HINGE generated very similar assemblies.

Benchmarking with the METAGENOME dataset. The METAGENOME dataset consists of Pacific Biosciences reads from a synthetic community of 20 bacteria. Since 3 of 20 bacterial genomes in the metagenomic sample had coverage below 1× (*Methanobrevibacter smithii, Candida albicans*, and *Streptococcus pneumoniae*), they were excluded from the benchmarking analysis. Since other assemblers performed poorly on the METAGENOME dataset, we limited our benchmarking to Flye and Canu, which assembled this dataset with NGA50 = 1,277 kb (84 misassemblies) and NGA50 = 1,061 kb (99 misassemblies), respectively (see Table 1). Supplementary Note 2 illustrates that most misassemblies in the METAGENOME dataset probably represent differences between the genomes in the METAGENOME sample and the reference genomes rather than real misassemblies.

Flye performed better than Canu for five genomes and Canu performed better that Flye for four genomes. In particular, Flye produced a better assembly of *Rhodobacter sphaeroides*, which has the lowest coverage (24×) among the 17 analyzed genomes (NGA50 = 2 Mb for Flye, compared with 54 kb for Canu). Comparison between the metagenome assemblies and the inferred isolate assemblies (from reads matched to the reference genomes) suggests that our metagenomics assemblies could be further improved by a better handling of datasets with uneven coverage.

Benchmarking with the YEAST dataset. The YEAST dataset contains PacBio and Oxford Nanopore Technology (ONT) reads from the *Saccharomyces cerevisiae* S288c genome of length 12.1 Mb at 30× coverage[22]. Similarly to the original study, we used the full set of ONT reads in the YEAST-ONT dataset (30× coverage) but downsampled the PacBio reads from the original 120× coverage to 30× in the YEAST-PacBio dataset to have their coverage distribution be similar to the ONT data. Assembling this dataset with the original 120× coverage results in better assemblies; for example, the NGA50 increased from 560 kb to 732 kb for the Flye assembly (Flye fully assembled 14 of 16 yeast chromosomes). Table 1 illustrates that all of the assemblers tested except HINGE produced YEAST-PacBio assemblies with similar NGA50 values ranging from 560 kb for Flye to 603 kb for Canu (HINGE resulted in a lower NGA50 of 361 kb). Flye generated the most accurate assembly with 5 errors (versus 13 errors for Canu). Although Miniasm generated an assembly with only ~90% sequence identity, Miniasm + ABruijn contigs had 99.93% accuracy. Canu and Flye resulted in assemblies with the highest sequence identity (above 99.95%).

The YEAST-ONT assemblies show a similar trend, with all assemblers except HINGE producing similar NGA50 values ranging from 637 kb (Falcon) to 723 kb (Miniasm). Flye generated the most accurate assembly with 9 errors (18 errors for Canu). Supplementary Fig. 2 shows the assembly graph generated by Flye.

Analyzing the WORM dataset. The WORM dataset contains PacBio reads from the *Caenorhabditis elegans* genome of length ~100 Mb at 40× coverage. Flye and Canu produced the most contiguous assemblies (NGA50 = 1,893 kb and 1,974 kb, respectively). However, Canu showed an increased number of misassemblies (190) compared with Flye (111) and Falcon (118). Flye was faster than Canu and Falcon in assembling the WORM dataset (128, 780, and 945 minutes of wall clock time, respectively (see Supplementary Note 1 for more details). With an increase in genome size, Flye achieves close to an order of magnitude speed-up as compared with Canu: for example, 140 versus 1,100 hours to assemble the *Drosophila melanogaster* genome. This speed-up highlights the advantages of skipping the time-consuming read-correction step and replacing conventional contig generation with the much more rapid generation of disjointigs. Supplementary Fig. 3 shows the assembly graph generated by Flye.

Since inferring the length of long tandem repeats is a difficult problem in short-read assembly, tandem repeats in many reference genomes might be misassembled. Supplementary Fig. 4 demonstrates that Flye improves on other long-read assemblers in reconstructing tandem repeats and reveals that some differences between the Flye assembly and the reference *C. elegans* genome probably represent differences with the reference rather than misassemblies by Flye.

Analyzing the HUMAN and HUMAN+ datasets. The HUMAN dataset contains ONT reads from the GM12878 human cell line at 30× coverage complemented by a set of short Illumina reads at 50× coverage. The HUMAN+ dataset combines the HUMAN dataset with a dataset of ultra-long ONT reads (those with reads N50 > 100 kb; that is, 50% of the total sequence data in reads longer than 100 kb) at 5× coverage[23]. Since Canu improved on Falcon and Miniasm in assembling large genomes[7], we only benchmarked Flye against Canu for the human genome datasets. The Canu HUMAN assembly was generated in ref. [23], and the assembly of the HUMAN+ dataset was later updated by the authors using the latest Canu 1.7 version. We also analyzed hybrid MaSuRCA assemblies of the HUMAN and HUMAN+ datasets[24], which are available from the MaSuRCA website.

Currently, the ONT assemblies have many base-calling errors (the Flye and Canu HUMAN assemblies had 1.2% and 2.8% error rate, respectively) because of the biased error pattern in ONT reads. Although the Nanopolish tool contributed to a reduction in the base-calling errors of the ONT assemblies[25], the resulting error rate

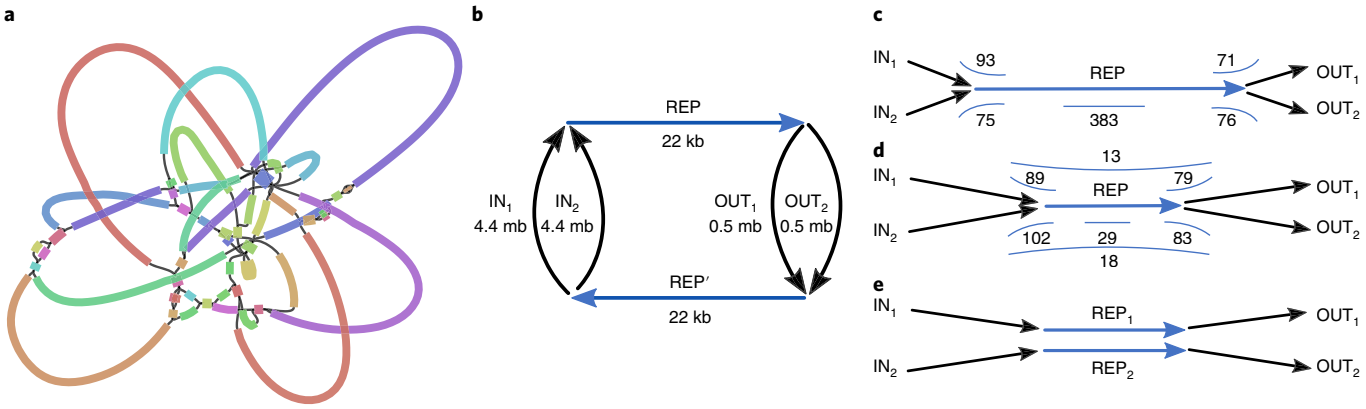

Fig. 3 | Resolving an unbridged repeat. a, An assembly graph of SMS reads from the *E. coli* strain EC9964 genome visualized with Bandage[30]. **b**, The untangled assembly graph (after resolving bridged repeats in the graph on the left) contains a single unbridged repeat REP (and its complement REP′) of length 22 kb. The incoming edges into the initial vertex (outgoing edges from the terminal vertex) of edge REP are denoted IN_1 and IN_2 (OUT_1 and OUT_2). Two complementary strands are fused together in a single connected component. It is unclear whether the genome traverses the assembly graph as $IN_1 \rightarrow REP \rightarrow OUT_1 \rightarrow REP'$ or as $IN_1 \rightarrow REP \rightarrow OUT_2 \rightarrow REP'$. **c**, A total of 93, 71, 75, and 76 reads traverse both IN_1 and REP, IN_2 and REP, REP and OUT_1, and REP and OUT_2, respectively. The span of 383 reads falls entirely within edge REP. **d**, After assigning 93 reads that traverse both IN_1 and REP to the first repeat copy, and 71 reads that traverse both IN_2 and REP to the second repeat copy, we 'move forward' into the repeat and construct two differing consensus sequences for a 8.6-kb-long prefix of REP with divergence 9.8% (two consensus sequences for a 6.8-kb-long suffix of REP when we 'move backward' into the repeat). The length of the repeat edge is reduced to $22.0 - 8.6 - 6.8 = 6.6$ kb, resulting in the emergence of $13 + 18 = 31$ spanning reads for this repeat, all of them supporting a *cis* transition (IN_1 with OUT_1 and IN_2 with OUT_2). **e**, Resolved instances of the repeat with consensus sequences REP_1 and REP_2 and divergence 6.9%.

Table 1 | Assembly statistics for the YEAST, WORM, HUMAN, and HUMAN+ datasets generated using QUAST.

Dataset	Assembler	Length (Mb)	No. contigs	NG50 (kb)	Reference coverage (%)	Reference percentage identity (%)	No. misassemblies	NGA50 (kb)
YEAST-PacBio	Flye	12.1	28	670	98.3	99.95	5	560
	Canu	12.4	33	708	99.5	99.95	13	603
	Falcon	12.1	42	562	97.5	99.81	27	562
	HINGE	12.2	45	440	91.9	98.81	19	361
	Miniasm + ABruijn	12.2	36	600	98.2	99.93	11	592
YEAST-ONT	Flye	12.1	28	810	98.7	99.04	9	660
	Canu	12.2	41	800	99.1	98.96	18	655
	Falcon	11.9	41	662	97.4	98.81	17	637
	HINGE	12.2	64	309	92.5	97.94	59	292
	Miniasm + ABruijn	11.6	24	723	98.8	99.03	12	723
WORM	Flye	103	85	3,256	99.5	99.93	111	1,893
	Canu	108	175	2,954	99.7	99.93	190	1,974
	Falcon	101	106	2,291	98.7	99.78	118	1,242
	HINGE	103	64	2,710	98.0	99.40	174	1,441
	Miniasm + ABruijn	108	178	2,314	99.6	99.93	181	1,437
HUMAN	Flye + Pilon	2,776	1,069	7,886	96.4	99.70	879	6,349
	Canu + Pilon	2,730	2,195	3,209	95.4	99.49	1,200	2,870
	MaSuRCA	2,768	1,269	4,670	95.1	99.84	1,500	3,812
HUMAN+	Flye + Pilon	2,823	782	18,181	97.0	99.69	1,487	11,800
	Canu + Pilon	2,815	798	10,410	96.8	99.81	1,455	7,007
	MaSuRCA	2,876	1,111	8,425	97.5	99.80	2,101	5,581

The NG50 of an assembly is the largest possible number *L*, such that all contigs of length *L* or longer cover at least 50% of the genome. Given an assembled set of contigs and a reference genome, a corrected assembly is formed by breaking each erroneously assembled contig at its breakpoints, resulting in shorter contigs[19]. The NGA50 of an assembly is defined as the NG50 of its corrected assembly. The minimum contig size was set to 5 kb for the YEAST and WORM assemblies and to 50 kb for the HUMAN assemblies. The human reference was modified by masking the low-complexity centromere regions of the chromosomes.

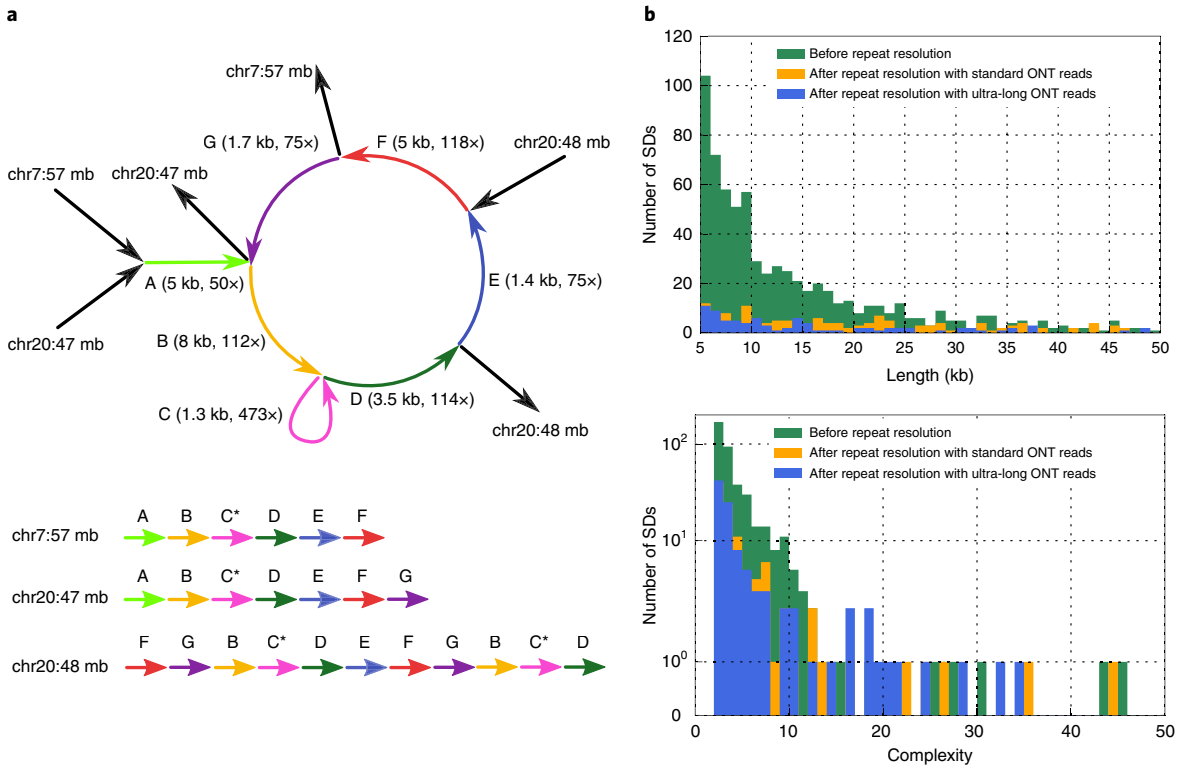

Fig. 4 | An SD from the Flye assembly of the HUMAN dataset and the distribution of the lengths and complexities of all SDs from the same assembly. **a**, A mosaic SD of complexity 7 represented as a connected component formed by repeat edges (7 colored edges of total length 25.7 kb) in the assembly graph of the HUMAN dataset (flanking unique edges shown in black). The loop-edge C with coverage 473× represents a tandem repeat C* with unit length 1.3 kb that is repeated ~19 times. The colored edges of the assembly graph align to a region on chromosome 7 of length 31 kb and two regions on chromosome 20 of lengths 30 kb and 46 kb. These three instances of SDs were not resolved using standard ONT reads but were resolved using ultra-long reads in a way that is consistent with the reference human genome. **b**, Statistics are given before resolving bridged repeats (green), after resolving bridged repeats with standard ONT reads (orange), and with ultra-long ONT reads (blue). Only SDs between 5 kb and 50 kb in length and with complexity between 2 and 50 contributed to the SD length and SD complexity histograms. Only two SDs have complexity exceeding 50 before bridged repeat resolution. Of the 688 SDs between 5 kb and 50 kb, 545 were resolved using the standard ONT reads, and ultra-long reads resolved an additional 58 SDs. There were 1,256 simple SDs before bridged repeat resolution and 143 after bridged repeat resolution with ultra-long reads. Since Flye usually resolves SDs shorter than the typical read length, the SDs identified by Flye do not include many known human SDs.

is still an order of magnitude higher than the error rates of Illumina or PacBio assemblies. Since most errors in the ONT assemblies are frameshift-introducing indels, they are particularly problematic for downstream applications.

To mitigate the high error rates of these ONT assemblies, we used Pilon[26] in the indel correction mode to polish Flye and Canu assemblies using Illumina reads. Although such polishing reduced the error rates (to 0.30% for Flye + Pilon and to 0.51% for Canu + Pilon), we note that Illumina-based read correction of ONT assemblies has limitations, especially for repetitive regions with low short-read mappability.

It turns out that Flye assembled a larger fraction of the human genome (96.4%) than Canu (95.4%) and MaSuRCA (95.1%). Interestingly, Flye and MaSuRCA, in contrast to Canu, assembled some difficult-to-assemble, low-complexity centromeric chromosome regions, which are hard to benchmark using reference-based methods. To provide a fair comparison between all three assemblers using QUAST, we thus modified the hg38 reference by masking the centromeric regions using the coordinates from the UCSC Genome Browser.

For the HUMAN dataset, Flye, MaSuRCA, and Canu generated assemblies with NGA50 values equal to 6.35 Mb (879 assembly errors), 3.81 Mb (1,500 assembly errors), and 2.87 Mb (1,200 assembly errors), respectively. The MaSuRCA assembly had a slightly higher percentage identity with the reference (99.84% compared with 99.70% for Flye + Pilon and 99.49% for Canu + Pilon).

For the HUMAN+ dataset, Flye, Canu, and MaSuRCA generated assemblies with NGA50 values equal to 11.8 Mb (1,487 assembly errors), 7 Mb (1,455 assembly errors), and 5.6 Mb (2,101 assembly errors), respectively. As expected, incorporating ultra-long ONT reads resulted in a more contiguous assembly for all assemblers.

SDs in the human genome. The repeat graph constructed by Flye reveals the complex mosaic structure of SDs. Flye classifies all edges in the graph into unique and repeat edges by analyzing how reads traverse the graph and by using coverage-based arguments (see Methods). After removing all unique edges from the assembly graph, only the connected components formed by repeat edges remain, which reveal the SDs encoded by the repeat edges in the graph. We define the complexity (length) of an SD as the number (total length) of edges in its connected component. Figure 4 (left) illustrates a mosaic SD of complexity 7 and length 25.7 kb (the 7 colored repeat edges form a connected component in the Flye assembly graph after removing all of the unique edges). An SD is classified as simple if its complexity is 1 and mosaic otherwise[14,15]. Figure. 4 (right) shows the distributions of lengths and complexities of SDs identified by Flye and illustrates the power of the assembly graph for repeat resolution.

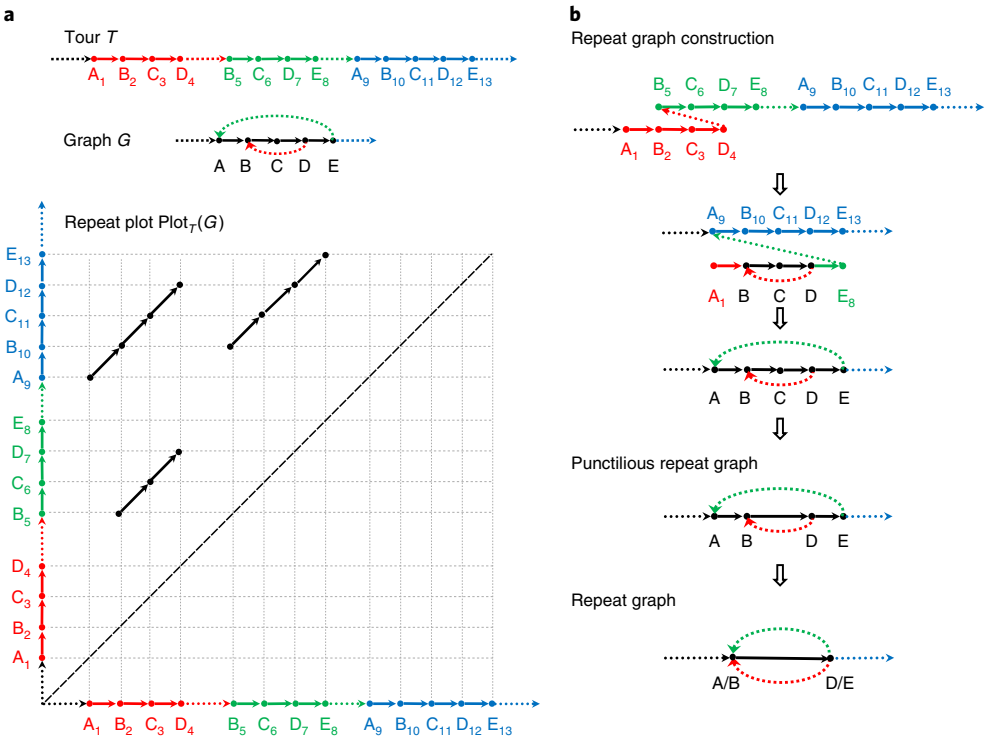

Fig. 5 | Constructing the repeat plot of a tour in the graph and constructing the repeat graph from a repeat plot. a, A tour $T = \ldots A_1B_2C_3D_4\ldots B_5C_6D_7E_8\ldots A_9B_{10}C_{11}D_{12}E_{13}\ldots$ in a graph G with red, green, and blue instances of a repeat that includes two copies of vertices A and E and three copies of vertices B, C, and D. Dots represent multiple vertices that appear before, between, and after these three instances of the repeat. The repeat plot $\text{Plot}_T(G)$ consists of three diagonals representing the three instances of the repeat in the tour. The trivial self-alignment of the entire genome against itself is shown by the main dotted diagonal (the points below this diagonal are not shown). Since vertex A in the graph is visited twice in tour T, it results in a single point (1, 9) in $\text{Plot}_T(G)$. Vertex B results in points (2, 5), (2, 10), and (5, 10); vertex C results in points (3, 6), (3, 11), and (6, 11); vertex D results in points (4, 7), (4, 12), and (7, 12); and vertex E results in the point (8, 13). **b**, Constructing the punctilious repeat graph from the repeat plot by gluing vertices with indices i and j for each point (i, j) in the repeat plot. Each non-branching path in the graph is substituted by a single edge with length equal to the number of edges in this path. The lengths of the short edges (A, B) and (D, E) in the resulting graph are equal to 1 and the length of the long edge (B, D) is equal to 2 (for edge length threshold $d = 1$). The punctilious repeat graph (second graph from the bottom) is transformed into the repeat graph (bottom-most graph) by contracting the short edges (A, B) and (D, E).

There are 1,748 repeat edges longer than 5 kb, forming 749 connected components in the Flye assembly graph of the HUMAN dataset before performing bridged repeat resolution. After bridged repeat resolution with ultra-long reads, there are only 765 repeat edges, forming 107 connected components in the assembly graph. Of these, 73 (34) represent mosaic (simple) SDs (most simple SDs represent isolated edges and loop-edges). See Supplementary Note 3 for more details.

A theoretical framework for the repeat graph construction. In addition to the described Flye algorithm, we provide a mathematical formulation of the repeat characterization problem and describe an alternative algorithm for repeat graph construction (Fig. 5). The Methods section provides additional details and explains the relation between the theoretical framework and the implementation in Flye.

Discussion

We describe the Flye algorithm for constructing an assembly graph from SMS reads and demonstrate that repeat characterization improves genome assembly. We show how to use the assembly graph to resolve unbridged repeats using variations between repeat copies and compared Flye with the Canu, Falcon, HINGE, Miniasm, and MaSuRCA assemblers.

In the case of the BACTERIA datasets, Flye and HINGE showed good agreement in the structure of constructed assembly graphs. Flye showed substantial improvement compared with HINGE on more complex eukaryotic datasets and generated the most accurate assemblies of the YEAST and WORM datasets; Flye and Canu also produced the best assembly contiguity in the case of the WORM dataset. For the more complex HUMAN and HUMAN+ datasets, Flye generated more contiguous and accurate assemblies than Canu and MaSuRCA, while being notably faster. Although assemblies of ONT reads feature rather high base-calling error rates (1.2% for the Flye HUMAN assembly), polishing the Flye assembly graph using Illumina reads has the potential to reduce the error rates by an order of magnitude.

The fact that Flye substantially improved on the Canu and MaSuRCA assemblies of the human genome suggests that there are still unexplored avenues for increasing the contiguity of SMS assemblies. We believe that better algorithms for resolving unbridged repeats in assembly graphs have the potential to greatly improve SMS assemblies, potentially increasing their NGA50 values by an order of magnitude. Flye constructed a repeat graph of the human genome with only 765 repeat edges representing various long SDs. Our algorithm for resolving unbridged repeats resolved only a small fraction of these SDs since it is currently limited to simple SDs (the vast majority of human SDs are mosaic). Moreover, it currently has difficulties resolving highly similar SDs, for example, SDs with less than 1% divergence. Although we reported the resolution of highly similar SDs on simulated datasets (as did a previous study[18]), most unbridged repeats resolved by Flye and Canu

are simple repeats with divergence exceeding 3%. Extending Flye to mosaic SDs and highly similar SDs has the potential to resolve most of the remaining unbridged repeats, since the vast majority of SDs in the human genome diverge by more than 1% (ref. [15]). Since there are only 53 long SDs (with length exceeding 15 kb) in the human genome that diverge by less than 1%, an SMS assembler that accurately resolves highly similar unbridged repeats will result in highly contiguous human genome assemblies, thus reducing the need for additional genome-finishing experiments (such as using Hi-C and/or optical maps).

Assembly graphs represent a special case of breakpoint graphs[27], and they are therefore well suited for analyzing structural variations[28,29] and SDs[14,15]. Flye assembly graphs provide a useful framework for reconstructing SDs and planning additional genome-finishing experiments.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, statements of data availability, and associated accession codes are available at https://doi.org/10.1038/s41587-019-0072-8.

Received: 14 May 2018; Accepted: 6 February 2019;
Published online: 1 April 2019

References

1. Koren, S. et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol.* **30**, 693–700 (2012).
2. Chin, C. S. et al. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat. Methods* **10**, 563–569 (2013).
3. Berlin, K. et al. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.* **33**, 623–630 (2015).
4. Chin, C. S. et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nat. Methods* **13**, 1050–1054 (2016).
5. Li, H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32**, 2103–2110 (2016).
6. Lin, Y. et al. Assembly of long error-prone reads using de Bruijn graphs. *Proc. Natl Acad. Sci. USA* **113**, E8396–E8405 (2016).
7. Kamath, G. M., Shomorony, I., Xia, F., Courtade, T. A. & David, N. T. HINGE: long-read assembly achieves optimal repeat resolution. *Genome Res.* **27**, 747–756 (2017).
8. 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).
9. Nowoshilow, S. et al. The axolotl genome and the evolution of key tissue formation regulators. *Nature* **554**, 50–55 (2018).
10. Ghurye, J., Pop, M., Koren, S., Bickhart, D. & Chin, C. S. Scaffolding of long read assemblies using long range contact information. *BMC Genomics* **18**, 527 (2017).
11. Weissensteiner, M. H. et al. Combination of short-read, long-read, and optical mapping assemblies reveals large-scale tandem repeat arrays with population genetic implications. *Genome Res.* **27**, 697–708 (2017).
12. Pevzner, P. A., Tang, H. & Tesler, G. De novo repeat classification and fragment assembly. *Genome Res.* **14**, 1786–1796 (2004).
13. Bankevich, A. et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.* **19**, 455–477 (2012).
14. Jiang, Z. et al. Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *Nat. Genet.* **39**, 1361–1368 (2007).
15. Pu., L., Lin, Y. & Pevzner, P. A. Detection and analysis of ancient segmental duplications in mammalian genomes. *Genome Res.* **28**, 901–909 (2018).
16. Bao, Z. & Eddy, S. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Res.* **8**, 1269–1276 (2002).
17. Schmid, M. D. et al. Pushing the limits of de novo genome assembly for complex prokaryotic genomes harboring very long, near identical repeats. *Nucleic Acids Res.* **46**, 8953–8965 (2018).
18. Tischler, G. Haplotype and repeat separation in long reads. Preprint at *bioRxiv* https://doi.org/10.1101/145474 (2017).
19. Mikheenko, A., Prjibelski, A., Saveliev, V., Antipov, D. & Gurevich, A. Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics* **34**, i142–i150 (2018).
20. Edmonds, J. & Johnson, E. L. Matching, Euler tours and the Chinese postman. *Math. Program.* **5**, 88–124 (1973).
21. Antipov, D., Korobeynikov, A., McLean, J. S. & Pevzner, P. A. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics* **32**, 1009–1015 (2015).
22. Giordano, F. et al. De novo yeast genome assemblies from MinION, PacBio and MiSeq platforms. *Sci. Rep.* **7**, 3935 (2017).
23. Jain, M. et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat. Biotechnol.* **36**, 338–345 (2018).
24. Zimin, A. V. et al. Hybrid assembly of the large and highly repetitive genome of *Aegilops tauschii*, a progenitor of bread wheat, with the MaSuRCA mega-reads algorithm. *Genome Res.* **27**, 787–792 (2017).
25. Simpson, J. T. et al. Detecting DNA cytosine methylation using nanopore sequencing. *Nat. Methods* **14**, 407 (2017).
26. Walker, B. J. et al. Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PloS ONE* **9**, e112963 (2014).
27. Lin, Y., Nurk, S. & Pevzner, P. A. What is the difference between the breakpoint graph and the de Bruijn graph? *BMC Genomics* **15**, S6 (2014).
28. Chaisson, M. J. P. et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature* **51**, 608–611 (2015).
29. Nattestad, M. S. et al. Complex rearrangements and oncogene amplifications revealed by long-read DNA 2 and RNA sequencing of a breast cancer cell line. *Genome Res.* **28**, 1126–1135 (2018).
30. Wick, R. R., Schultz, M. B., Zobel, J. & Holt, K. E. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* **31**, 3350–3352 (2015).

Acknowledgements

We are indebted to S. Nurk for his multiple rounds of critique and suggestions that have improved the paper. We are also grateful to A. Mikheenko, B. Behsaz, L. Pu, and G. Tesler for their comments. This work is supported by NSF/MCB-BSF grant no. 1715911.

Author contributions

All authors contributed to developing the Flye algorithms and writing the paper. M.K., Y.L., and J.Y. implemented the Flye algorithm. M.K. benchmarked Flye and other assembly tools. P.A.P. directed the work.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41587-019-0072-8.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to P.A.P.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Methods

Repeat characterization problem. Below we describe the abstract repeat characterization problem and explain how it relates to genome assembly. Consider a tour $T = v_1, v_2, \ldots v_n$ of length n visiting all vertices of a directed graph G. We say that the ith and jth vertices in the tour T are equivalent if they correspond to the same vertex of the graph, that is, $v_i = v_j$. The set of all pairs of equivalent vertices forms a set of points (i, j) in a two-dimensional grid that we refer to as the repeat plot $\text{Plot}_T(G)$ of the tour T (Fig. 5). The transformation of a tour T traversing a known graph G into the repeat plot $\text{Plot}_T(G)$ is a simple procedure. Below, we address the reverse problem that is at the heart of genome assembly, repeat characterization and synteny block construction: given an arbitrary set of points Plot, in a two-dimensional grid, find a graph $G = G(\text{Plot})$ and a tour T in this graph such that $\text{Plot} = \text{Plot}_T(G)$.

A dot-plot of a genome is a matrix that graphically represents all repeats in a genome[31]. In the case of repeat characterization, we are interested in the dot-plot Plot formed by non-overlapping alignment-paths representing all high-scoring local self-alignments of a genome against itself (below, we refer to these alignments as simply self-alignments). Each self-alignment reveals two instances of a repeat corresponding to contiguous segments x and y in the genome (x and y are called the spans of the alignment). Given a genome of length n and a set of its self-alignments Plot, the repeat characterization problem amounts to constructing a graph G and a tour T of length n in this graph (each segment of the genome corresponds to a subpath of the graph traversed by the tour) such that $\text{Plot} = \text{Plot}_T(G)$ and the tour T is alignment-compatible. A tour is alignment-compatible with respect to the dot-plot Plot if, for each alignment with spans x and y in Plot, paths in the graph corresponding to segments x and y coincide.

Generating the repeat plot of a genome. Our goal is to construct both the repeat graph of a genome and an alignment-compatible tour in this graph. Constructing the de Bruijn graph of a genome based on long k-mers will not solve this problem since the differences between imperfect repeat copies mask the repeat structure of the genome. Constructing the de Bruijn graph based on short k-mers will not solve this problem due to the presence of repeating short k-mers within long repeats (these k-mers lead to a tangled repeat graph). Thus, at the initial stage, Flye generates all self-alignments (repeats) of a genome and combines them into a repeat plot Plot. However, it is unclear how to solve the reverse problem of generating the repeat graph $G(\text{Plot})$ of the genome.

To address this problem for a 'genome' representing a concatenate of accurate short reads, a previous study[12] described various graph simplification procedures, for example, bubble and whirl removals, that are now at the heart of various short-read assemblers such as SPAdes[13]. However, it is not clear how to generalize these procedures to make them applicable to error-prone SMS reads. Below, we show how to modify the concept of a punctilious repeat graph[12] so that it can be applied to assembling SMS reads.

Constructing a punctilious repeat graph. Let Alignments = Alignments(Genome, minOverlap) be the set of all sufficiently long (of length at least 'minOverlap') self-alignments of a genome 'Genome'. Flye sets the 'minOverlap' parameter as the N90 of the read-set (the N90 of reads is the largest possible number N such that all reads of length N or longer have a total length of at least 90% of the total sequence; 'minOverlap' varies from 3,000 to 5,000 nucleotides for the SMS datasets analyzed in this paper).

Given a set of self-alignments 'Alignments' of a genome 'Genome', we construct the punctilious repeat graph RepeatGraph(Genome, Alignments) by representing 'Genome' as a path consisting of |Genome| vertices (Fig. 5) and by 'gluing' each pair of vertices (positions in the genome) that are aligned against each other in one of the alignments in 'Alignments'[12]. Gluing vertices v and w amounts to substituting them by a single vertex that is connected by edges to all vertices that either vertex v or vertex w was connected to. We consider branching vertices (that is, vertices with either in-degree or out-degree differing from 1) in the resulting graph and substitute each non-branching path between them by a single edge of length equal to the number of original edges in this path. Edges in the punctilious repeat graph are classified as long (longer than a predefined threshold d with default value 500 nucleotides) and short (Fig. 5).

The punctilious repeat graphs of real genomes are very complex due to various artifacts[12,14]. For example, the starting/ending points of alignment-paths corresponding to three repeat copies starting at positions x, y, and z in the genome hardly ever start at points (x, y), (x, z), and (y, z) in the repeat plot. Because each repeat with m copies in the genome results in $\binom{m}{2}$ pair-wise alignments and each of the corresponding $\binom{m}{2}$ alignment-paths may have unique starting (ending) vertices that differ from all other starting/ending positions, there will be many gluing operations for the starting (ending) positions of this repeat. Note that each of these operations may form a new branching vertex in the punctilious repeat graph. For example, gluing the endpoints of the three diagonals in Fig. 5 results in the branching vertices A, B, D, and E in the graph. Punctilious repeat graphs of real genomes often contain many branching vertices, making it difficult to compactly represent repeats. We address this challenge by transforming the punctilious repeat graph into a simpler graph.

From punctilious repeat graph to repeat graph. As described before, the endpoints of alignment-paths representing the same repeat might not be coordinated among all pair-wise alignments of this repeat. These uncoordinated alignments result in a complex repeat graph with an excessive number of branching vertices and many short edges (shorter than a threshold d). The repeat graph RepeatGraph(Genome, Alignments, d) is defined as the result of contracting all short edges in the punctilious repeat graph (Fig. 5). The contraction of an edge is the gluing of the endpoints of this edge, followed by the removal of the loop-edge resulting from this gluing. Since the genome represents a tour visiting all edges in the repeat graph, we define the multiplicity of an edge in the repeat graph as the number of times this edge is traversed in the tour. Edges of multiplicity 1 are called unique edges and all other edges are called repeats.

Approximate repeat graphs. The described approach, although simple in theory, results in various complications in the case of real genomes, particularly in the case of inconsistent pair-wise alignments (see Supplementary Note 5). In the case of short reads, various graph simplification procedures[12,13] result in a modified repeat graph that represents a more sensible repeat characterization, but sacrifice the fine details of some repeats in favor of revealing the mosaic structure shared by different repeat copies. However, in the case of SMS assemblies, repeat graph (and A-Bruijn graph) construction results in excessively complex graphs that make the previously proposed graph simplification algorithm for A-Bruijn graph construction[12] inefficient and make it difficult to select sensible parameters for graph simplification. For example, it is unclear how to select an adequate 'bubble_size' parameter for bubble removal (small values of this parameter result in complex A-Bruijn graphs while large values result in oversimplified A-Bruijn graphs). While there exists a 'sweet spot' for this parameter in short-read assembly, we were not able to find such a spot for long-read assembly. That is why we departed from the original A-Bruijn graph framework and opted to construct a different version of the repeat graph (called the approximate repeat graph) based only on the endpoints of diagonals in the genomic dot-plot rather than the entire diagonals as in a previous study[12]. This approach led to a great reduction in running time and allowed us to bypass the bubble/whirl-removal steps (and the challenge of choosing parameters for these operations) altogether.

Some branching vertices in the repeat graph arise from the contraction of multiple vertices in the punctilious repeat graph; for example, vertices A and B were contracted into a single vertex A/B in the repeat graph in Fig. 5. Consider the set of all vertices in the punctilious repeat graph that gave rise to branching vertices in the repeat graph (vertices A, B, D, and E in Fig. 5) and let Breakpoints = Breakpoints(Genome, Alignments, d) be the set of all positions in the genome that gave rise to these vertices (Breakpoints = {1, 2, 4, 5, 7, 8, 9, 10, 12, 13} in Fig. 5). This set of vertices forms a set of short, contiguous genomic segments (segments (1, 2), (4, 5), (7–10), and (12, 13) in Fig. 5) that contain all horizontal and vertical projections of the endpoints of all alignments in 'Alignments'.

Flye approximates the set 'Breakpoints' by recruiting all horizontal and vertical projections of the endpoints of alignments from 'Alignments' to the main diagonal in the repeat plot. Figure 2 presents three alignments, resulting in eight projected points on the main diagonal. Two alignment endpoints are close if either of their projections on the main diagonal are located within a distance threshold d (including the case when a vertical projection of one endpoint coincides with or is close to a horizontal projection of another endpoint).

Flye clusters close endpoints together based on single linkage clustering. Applying this procedure (with $d = 0$) to eight breakpoints (projected endpoints) in Fig. 2 results in three clusters (breakpoints in the same cluster are painted with the same color). Figure 2 illustrates that gluing breakpoints that belong to the same clusters (and further collapsing parallel edges) results in an approximate repeat graph of the genome. However, although this procedure led to the correct repeat graph in the simple case shown in Fig. 2, the approximate repeat graph constructed based on the clustering of closely located breakpoints may differ from the repeat graph constructed based on the punctilious repeat graph. Supplementary Note 6 illustrates that mosaic repeats and inconsistencies of local alignments may result in an 'incorrect' clustering-based repeat graph. Below, we explain how Flye extends the set 'Breakpoints to address this complication.

Extending the set of breakpoints. As described above, Flye constructs the initial set 'Breakpoints' by projecting all endpoints of the alignments (in the set of self-alignments 'Alignments') onto the main diagonal in the repeat plot. Each point in an alignment-path in the |Genome| × |Genome| grid has two projections (horizontal and vertical) on the main diagonal. Note that projections of some internal points in an alignment-path may belong to 'Breakpoints'; for example, both projections of the middle point of the longest alignment-path in Fig. 2 (shown in purple) belong to 'Breakpoints'. Such internal points should be reclassified as new alignment endpoints (by breaking the alignment-path into two parts) to avoid inconsistencies during the construction of the repeat graph. However, for some internal points, only one of their two projections belongs to 'Breakpoints', leading to complications in the path-breaking process. Below, we explain how to break the alignment-paths into subpaths (and, at the same time, extend the set 'Breakpoints') to address this complication.

A point in an alignment-path is called valid if both of its projections belong to 'Breakpoints', and invalid if only one of its projections belongs to 'Breakpoints'. A set 'Breakpoints' is called valid if all points in all alignment-paths are valid, and invalid otherwise. In the case that the constructed set 'Breakpoints' is invalid, our goal is to add the minimum number of points to this set to make it valid. See Supplementary Note 6 for an example of an invalid point and a discussion on the importance of extending the set 'Breakpoints' to make it valid.

Flye iteratively adds the missing projection for each invalid point to the set 'Breakpoints' on the main diagonal until there are no invalid points left. Afterwards, it combines close points in 'Breakpoints' into segments using single linkage clustering (as described above). The set of resulting segments (defined by their minimal and maximal positions on the main diagonal) forms a set 'BreakpointSegments'. Two segments from 'BreakpointSegments' are equivalent if there exists a point in one of the alignment-paths such that one of its projections on the main diagonal falls into the first segment and another falls into the second segment.

Each repeat of multiplicity m typically corresponds to m segments in 'BreakpointSegments' corresponding to m starting positions of this repeat in the genome (and the same number of segments corresponding to its ending positions). Note that the number of breakpoint segments resulting from this repeat is reduced as compared with the number of breakpoints, which can be as large as $\binom{m}{2}$ for the starting positions of the repeat (and the same number for its ending positions). Flye takes advantage of this reduction by selecting middle points of each breakpoint segment and only gluing these middle points rather than all breakpoints. Essentially, it redefines the endpoints of each alignment-path as the middle points of corresponding breakpoint segments.

Specifically, Flye constructs the approximate repeat graph by generating the set 'BreakpointSegments', selecting a middle point from each segment in 'BreakpointSegments', and gluing the two middle points for every pair of equivalent segments. Afterwards, it glues together parallel edges (edges that start and end at the same vertices) if the genome segments corresponding to these edges are aligned in 'Alignments', that is, if there exists an alignment with its x- and y-spans overlapping both these segments. For brevity, below we refer to the approximate repeat graph resulting from this procedure simply as the repeat graph.

From the repeat graph of a genome to the assembly graph of contigs. The ABruijn assembler[6] constructs a set of contigs but stops short of constructing the repeat graph of a genome based on these contigs (Supplementary Note 7 describes the challenge of assembling contigs into a repeat graph). The contig construction in ABruijn essentially amounts to finding extension reads for extending paths in the (unknown) repeat graph of the genome. Each extension read increases the length of the growing path until the extension process becomes ambiguous, that is, when it reaches a branching vertex in the (unknown) repeat graph. Afterwards, ABruijn decides whether to continue or to stop the path extension to avoid assembly errors. Since ABruijn does not know the exact locations of branching vertices, it uses the last extension read to extend the path beyond the branching vertex by at least 'minOverlap' nucleotides. As a result, each linear contig constructed by ABruijn satisfies the overhang property: it extends by at least *minOverlap* nucleotides before the first branching vertex and after the last branching vertex it traverses. Note that the same 'minOverlap' value is used during repeat graph construction.

Constructing disjointigs. ABruijn and other existing SMS assemblers invest substantial time into making sure that generated contigs are correctly assembled (represent subpaths of the genomic tour in the repeat graph). In contrast to ABruijn, Flye does not attempt to construct accurate contigs at the initial assembly stage but instead generates disjointigs as arbitrary paths in the (unknown) repeat graph of the genome. However, it constructs an accurate repeat graph (assembly graph) from error-prone disjointigs.

Flye randomly walks in the (unknown) assembly graph to generate random paths from this graph. Each non-chimeric read from 'Reads' defines a subpath of a genomic tour in an assembly graph. Flye extends this path by switching from the current read to any other overlapping read (with sufficiently long common jump-subpath) rather than a carefully chosen overlapping read[6], avoiding a time-consuming test to check whether this selection triggers an assembly error.

Since the resulting FlyeWalk algorithm (see Supplementary Note 8) does not invoke the contig correctness check, it constructs paths (chains of overlapping reads) that do not necessarily follow the genome tour through the assembly graph. Although it may appear counter-intuitive that inaccurate disjointigs constructed by FlyeWalk result in an accurate assembly graph, note that inaccurate paths (disjointigs) in the de Bruijn graph (a special case of the assembly graph) certainly result in an accurate assembly graph. Indeed, an assembly graph constructed from arbitrary paths in a de Bruijn graph is the same as the original de Bruijn graph (as long as these paths include all k-mers from the assembly graph). See Supplementary Note 9 for additional details.

Constructing the assembly graph from disjointigs. Similarly to ABruijn, Flye generates disjointigs satisfying the overhang property, which, as will be explained below, represents an important condition for constructing the repeat graph. Flye further concatenates all disjointigs (separated by delimiters) in an arbitrary order into a single string *Concatenate*. It further uses the longest jump-subpath approach[6] to generate the set 'Alignments' of all sufficiently long self-alignments within the resulting concatenate and constructs the assembly graph as the repeat graph of the concatenate RepeatGraph(Concatenate, Alignments, d).

It has been shown that the repeat graph of concatenated accurate reads (when alignments between reads do not extend beyond delimiters in the concatenate of all reads) approximates the repeat graph of the genome[12]. Supplementary Note 10 demonstrates that the assembly graph constructed from inaccurate disjointigs also approximates the repeat graph of the genome.

Figure 3 (left) presents the assembly graph of the SMS reads from an *Escherichia coli* genome. Flye further untangles this graph into a graph with just six edges (Fig. 3, middle) as described below.

Resolving bridged repeats in the assembly graph. Flye aligns all reads to the constructed assembly graph (see Supplementary Note 11) and uses them to identify the repeat edges in this graph (see Supplementary Note 12). It further transforms the assembly graph into the condensed assembly graph by contracting all of its repeat edges. Aligning a read to the assembly graph induces its alignment to the condensed assembly graph, and we focus on bridging reads that align to multiple edges in the condensed assembly graph. Untangling incident edges $e=(w, v)$ and $f=(v, u)$ in the condensed assembly graph amounts to substituting them by a single edge (w, u). Below, we describe how Flye uses bridging reads to untangle the condensed assembly graph and how this untangling contributes to resolving repeats in the assembly graph.

A bridging read in the condensed assembly graph is called an (e, f)-read if it traverses two consecutive edges e and f in this graph. For each pair of incident edges e and f in the condensed assembly graph, we define 'transition(e, f)' as the number of (e, f)-reads plus the number of (f', e')-reads, where e' and f' are complementary edges for e and f, that is, edges representing a complementary strand.

Given a set of bridging reads in the condensed assembly graph, we construct a transition graph as follows. Each edge e in the condensed assembly graph corresponds to vertices e^h and e^t in the transition graph, representing the head (start) and tail (end) of e, respectively. A complementary edge for e corresponds to the same vertices, but in the opposite order. Each (e, f)-read defines an undirected edge between e^t and f^h in the transition graph with weight equal to transition(e, f).

Note that the transition graph is bipartite for the simple case when the two subgraphs of the condensed assembly graphs, corresponding to complementary strands, do not share vertices. However, it is not necessarily bipartite in the case of genomes that contain long inverted repeats. Flye thus applies Edmonds' algorithm[32] to find a maximum weight matching in the transition graph and uses this matching for untangling the condensed assembly graph. For each edge (e^t, f^h) in the constructed matching, Flye additionally checks the confidence of the transition between edges e and f (see Supplementary Note 13 for details) and untangles e and f for each edge (e^t, f^h) in the transition graph that passes this check. Flye iteratively untangles edges in the condensed assembly graph and performs the corresponding iterative repeat resolution in the assembly graph.

Note that consecutive edges e and f in the condensed assembly graph are not necessarily consecutive in the assembly graph. Thus, after Flye untangles e and f, it uses one of the bridging (e, f)-reads to fill the gap between the end of e and the start of f in the assembly graph. Afterwards, most repeat edges in the assembly graph either represent long unbridged repeat edges (that are not bridged by any reads) or form paths consisting of repeat edges with total lengths typically exceeding the median read length.

Resolving unbridged repeats in the assembly graph. Flye takes advantage of the small variations between different repeat copies to resolve unbridged repeats. It identifies the variations between repeat copies, matches each read with a specific repeat copy using these variations, and uses these matched reads to derive a distinct consensus sequence for each repeat copy. The success of this approach is contingent on the presence of a sufficiently large number of variations between the different repeat copies. Therefore, the first step is to estimate the number and positions of variations between the repeat copies and to calculate the divergence of the various repeat copies from reads alone. Supplementary Note 14 describes how Flye calculates the divergence between repeat copies. The current version of Flye is limited to resolving unbridged repeats of multiplicity two in both haploid (for example, bacterial) and diploid (for example, human) genomes.

The idea of the algorithm is to assign each read to a specific repeat copy and then use the assigned reads to derive a distinct consensus sequence for each repeat copy. Figure 3 shows an example in which the 93 reads that traverse edges IN_1 and REP can be assigned to one repeat copy and the 75 reads that traverse edges IN_2 and REP can be assigned to another repeat copy. However, it is unclear how to assign other reads mapping to the edge REP to a specific repeat copy. Flye uses reads starting in the incoming edges (93 and 75 reads in Fig. 3) to 'move forward' into the repeat and construct two different prefixes of the repeat REP corresponding to the two copies of the repeat. In parallel, it uses reads ending in the outgoing edges (71 and 76 reads in Fig. 3) to 'move backward' into the repeat and construct two different suffixes of this repeat.

In each iteration of the algorithm, reads are assigned to a specific repeat copy, and then all of the reads assigned to each repeat copy are used to construct a consensus sequence for that copy. Thus, as the algorithm proceeds, more reads are assigned to specific repeat copies and the consensus sequence for each repeat copy grows longer. The algorithm terminates when no new reads can be assigned to read copies and the consensus sequences stop growing in length. There are two goals: to obtain distinct consensus sequences for each repeat copy and to determine the correct pairings of incoming and outgoing edges for each repeat copy.

Supplementary Note 15 describes each successive iteration of the algorithm in detail. Supplementary Note 16 evaluates its accuracy on simulated data, and Supplementary Table 2 provides information about Flye's performance on the unbridged repeats from the BACTERIA dataset.

Reporting Summary. Further information on research design is available in the Nature Research Reporting Summary linked to this article.

Data availability

All described datasets are publicly available through the corresponding repositories. The supplementary files, including the assemblies generated by Flye, are available at https://doi.org/10.5281/zenodo.1143753; NCTC PacBio reads: http://www.sanger.ac.uk/resources/downloads/bacteria/nctc/; PacBio metagenome dataset: https://github.com/PacificBiosciences/DevNet/wiki/Human_Microbiome_Project_MockB_Shotgun; PacBio *C. elegans* dataset: https://github.com/PacificBiosciences/DevNet/wiki/C.-elegans-data-set; PacBio/ONT *S. cerevisiae* dataset: https://github.com/fg6/YeastStrainsStudy. The ONT reads from the HUMAN and HUMAN+ datasets are available at https://github.com/nanopore-wgs-consortium/NA12878. The matching Illumina reads are available as SRA project ERP001229. The Canu HUMAN+ assembly was downloaded from https://genomeinformatics.github.io/na12878update. MaSuRCA assemblies are available from http://masurca.blogspot.com/.

Code availability

The Flye code used in this study is available in the online version of the paper. The most recent Flye version is freely available at http://github.com/fenderglass/Flye.

References

31. Gibbs, A. J. & McIntyre, G. A. The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences. *Eur. J. Biochem.* **16**, 1–11 (1970).
32. Edmonds, J. Paths, trees, and flowers. *Canad. J. Math.* **17**, 449–467 (1965).

nature research

Corresponding author(s): Pavel Pevzner

Last updated by author(s): Jan 8, 2019

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see Authors & Referees and the Editorial Policy Checklist.

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed
☒	☐ The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☒	☐ A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☒	☐ The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☒	☐ A description of all covariates tested
☒	☐ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☒	☐ A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐ For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☒	☐ Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated

Our web collection on statistics for biologists contains articles on many of the points above.

Software and code

Policy information about availability of computer code

Data collection	No custom software were used for data collection.
Data analysis	The manuscript presents a genome assembly algorithm called Flye. The code is available in the online version of the paper and at https://github.com/fenderglass/Flye. Additional data, such as genome assemblies that were generated in our analysis are available at https://doi.org/10.5281/zenodo.1143752. List of software used in the manuscript: Flye 2.3.5 (commit 20afeda), Canu 1.7.1 (commit dfa60b8), Falcon 0.3.0 (FALCON-Integrate commit 7498ef9), HINGE 0.5.0 (commit 79fdf66), Miniasm 0.2-r168-dirty (commit 40ec280) / Minimap2 2.8-r711 (commit 8fc5f8d), QUAST 5.0.0 (commit de6973bb), Graphviz 2.38.0, Bandage 0.8.1

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors/reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

All described datasets are publicly available through the corresponding repositories:
* The supplementary files, including the assemblies generated by Flye, are available at https://doi.org/10.5281/zenodo.1143753
* NCTC PacBio reads: http://www.sanger.ac.uk/resources/downloads/bacteria/nctc/.
* PacBio metagenome dataset: https://github.com/PacificBiosciences/DevNet/wiki/Human_Microbiome_Project_MockB_Shotgun.
* PacBio C. elegans dataset: https://github.com/PacificBiosciences/DevNet/wiki/C.-elegans-data-set
* PacBio / ONT S. cerevisiae dataset: https://github.com/fg6/YeastStrainsStudy

* The ONT reads from the HUMAN/HUMAN+ datasets are available at: https://github.com/nanopore-wgs-consortium/NA12878. The matching Illumina reads are available as SRA project ERP00122. The Canu HUMAN+ assembly was downloaded from: https://genomeinformatics.github.io/na12878update. MaSuRCA assemblies are available from: http://masurca.blogspot.com/

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	Does not apply, since the study does not include any statistical analysis
Data exclusions	No data were excluded from anlyses
Replication	Does not apply, since the study describes deterministic algorithms and does not include statistical analysis. We have references all publicly available datasets and software versions to ensure reproducibility of our analysis.
Randomization	Does not apply since our study does not involve data acquisition.
Blinding	Does not apply since our study does not require case/control comparison

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology
☒	☐ Animals and other organisms
☒	☐ Human research participants
☒	☐ Clinical data

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging