nature biotechnology

SUPPLEMENTARY INFORMATION

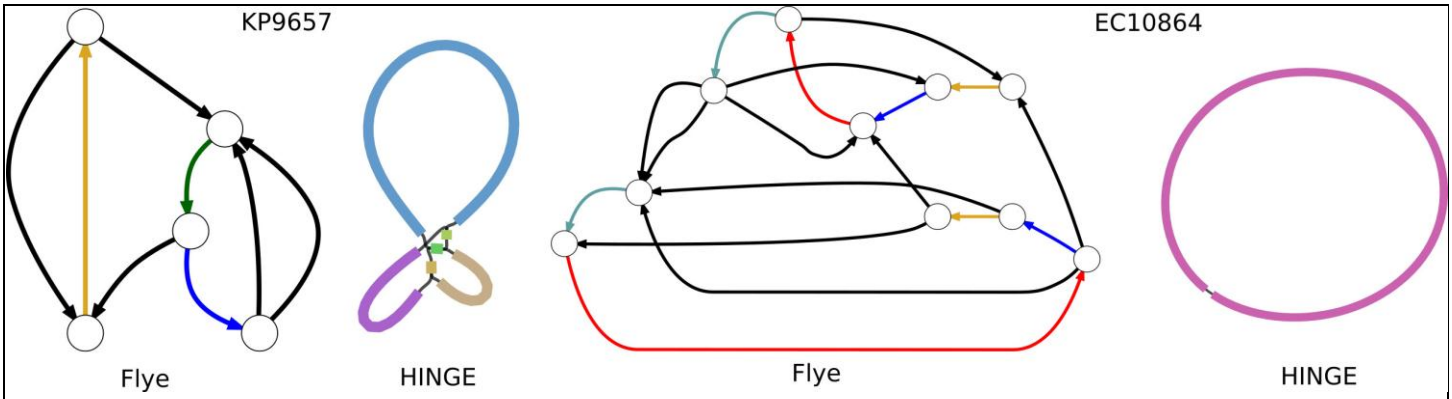
https://doi.org/10.1038/s41587-019-0072-8

In the format provided by the authors and unedited.

Assembly of long, error-prone reads using repeat graphs

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

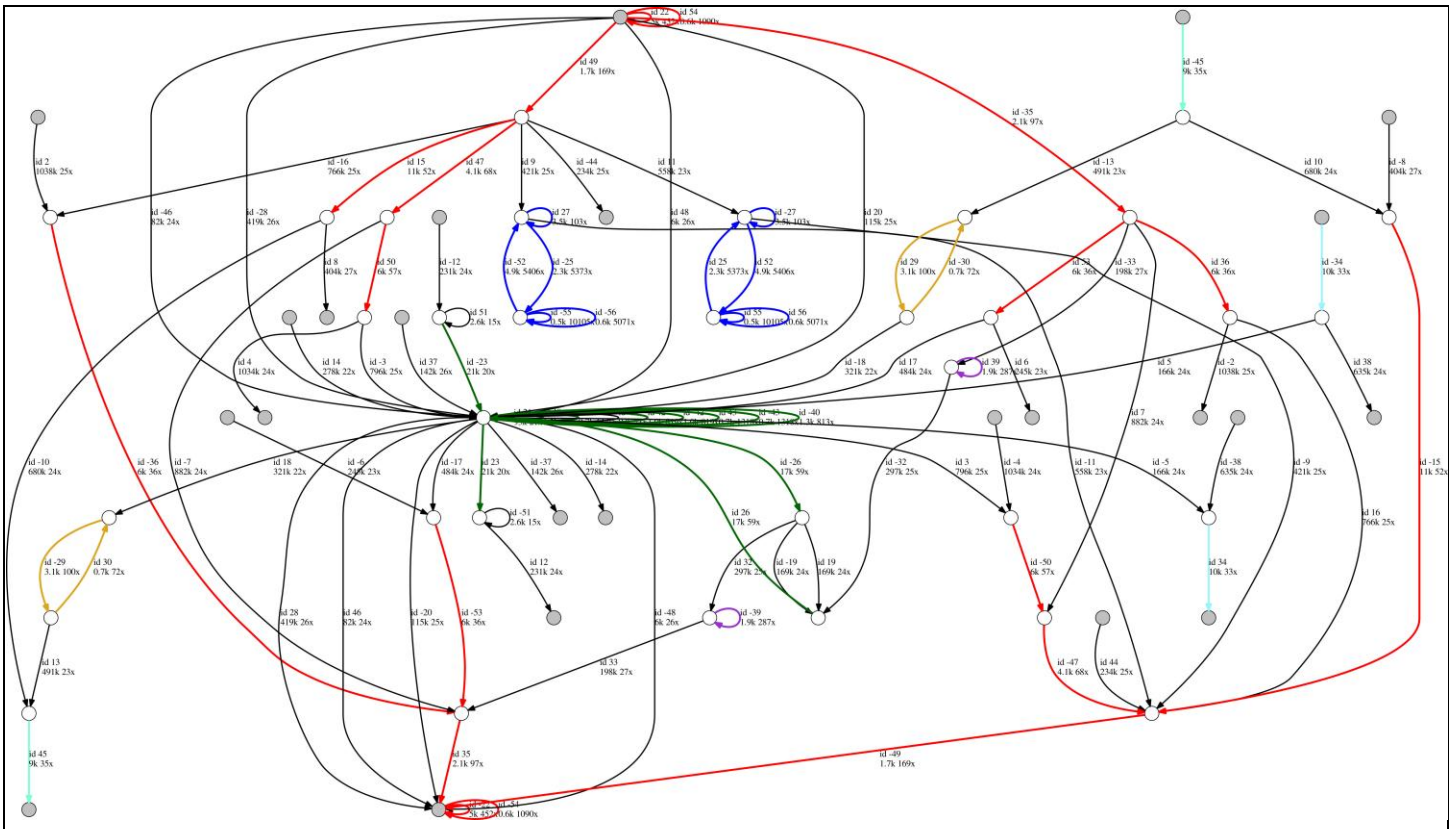
[1]Department of Computer Science and Engineering, University of California, San Diego, CA, USA. [2]Graduate Program in Bioinformatics and Systems Biology, University of California, San Diego, CA, USA. [3]Research School of Computer Science, Australian National University, Canberra, Australian Capital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



Supplementary Figure 1

A comparison of Flye and HINGE assembly graphs on bacterial genomes from the BACTERIA dataset.

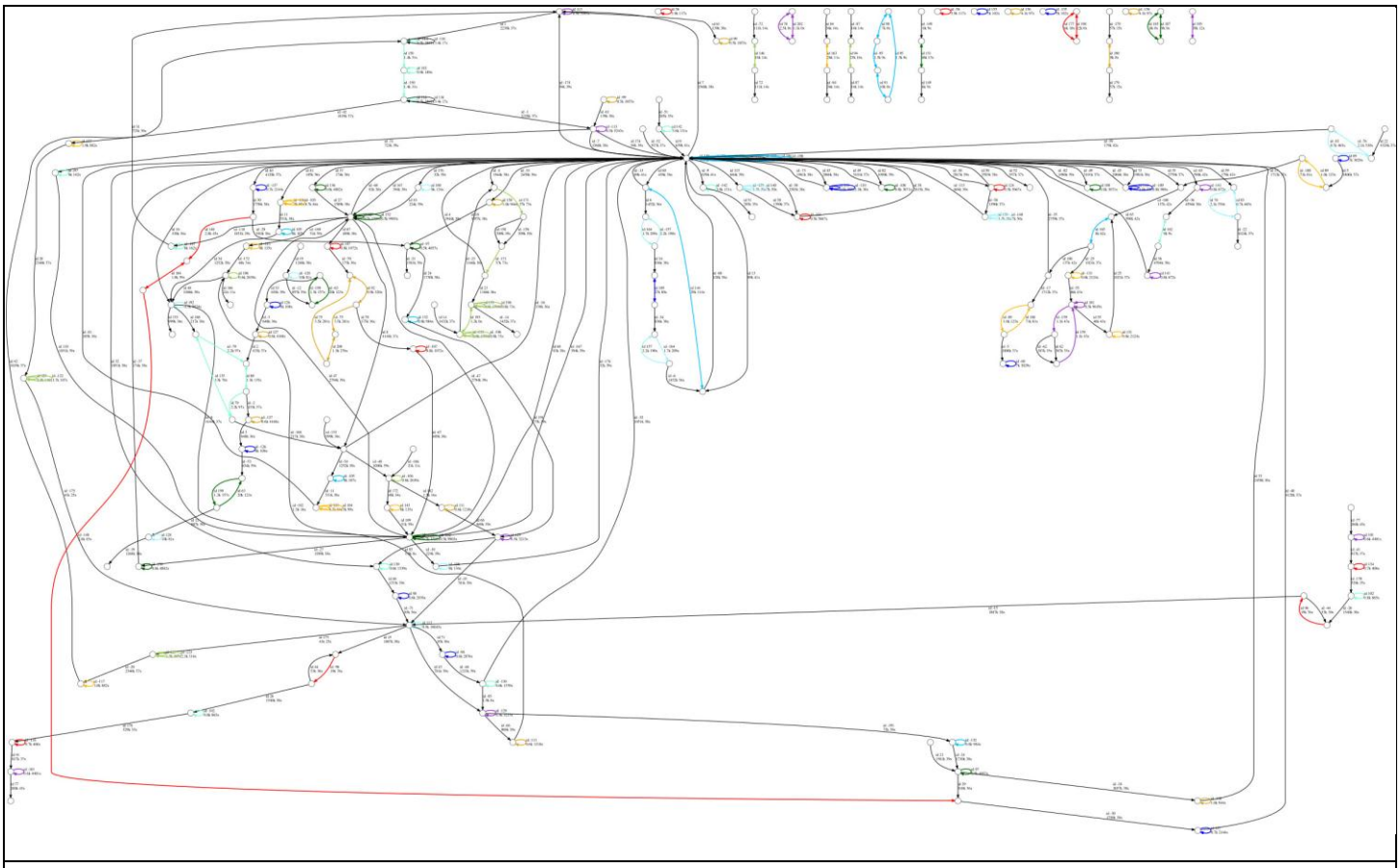
(Left) Flye and Hinge assembly graphs of the KP9657 dataset. There is a single unique edge entering into (and exiting) the unresolved "yellow" repeat and connecting it to the rest of the graph. Thus, this repeat can be resolved if one excludes the possibility that it is shared between a chromosome and a plasmid. In contrast to HINGE, Flye does not rule out this possibility and classifies the yellow repeat as unresolved. (Right) The Flye and Hinge assembly graphs of the EC10864 dataset show a mosaic repeat of multiplicity four formed by yellow, blue, red and green edges (the two copies of each edge represent complementary strands). HINGE reports a complete assembly into a single chromosome.



Supplementary Figure 2

The assembly graph of the YEAST-ONT dataset.

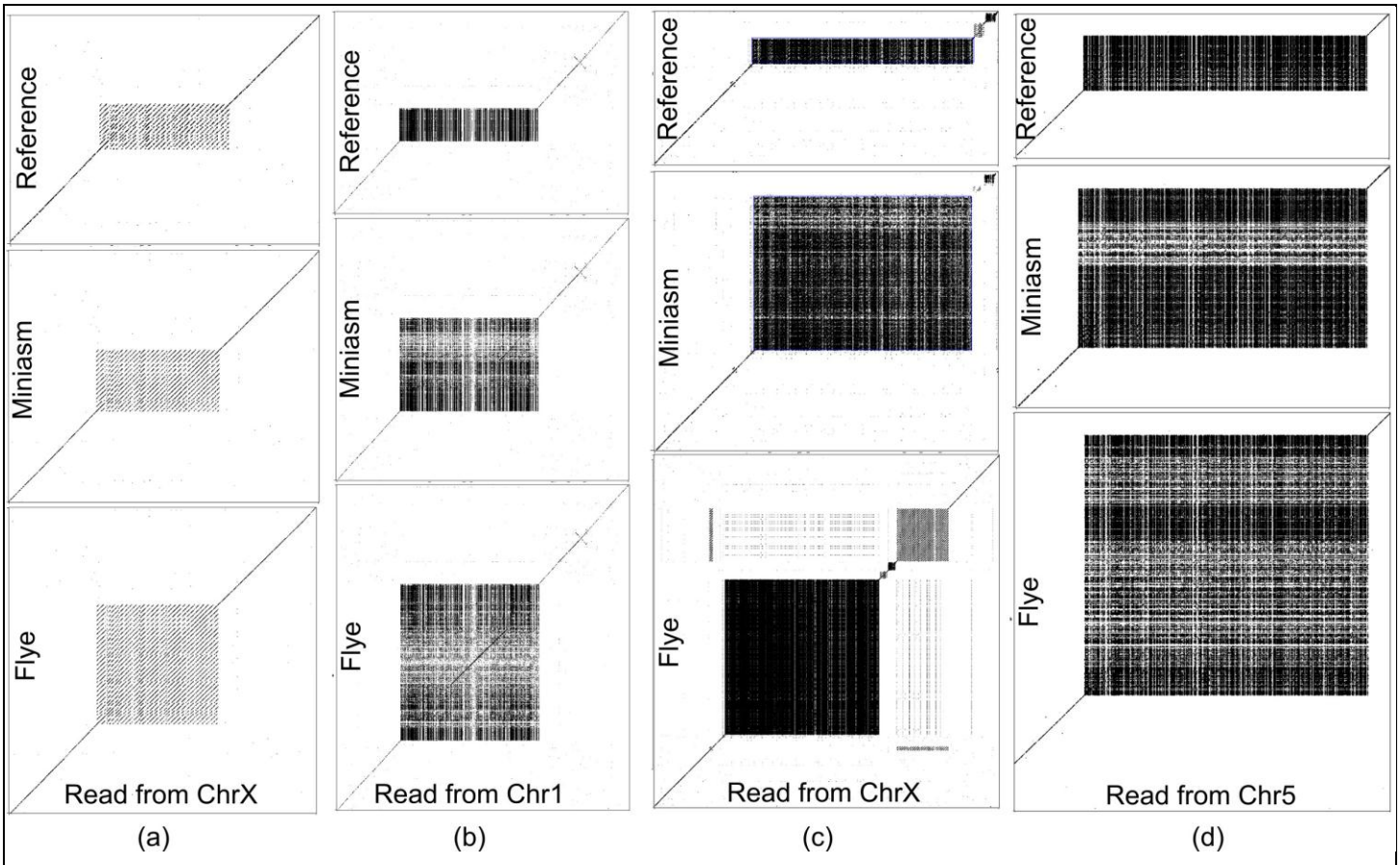
Edges that were classified as repetitive by Flye are shown in color, while unique edges are black. Flye assembled the YEAST-ONT dataset into a graph with 21 unique and 34 repeat edges and generated 21 contigs as unambiguous paths in the assembly graph. A path $v_1, \ldots v_i, v_{i+1} \ldots v_n$ in the graph is called unambiguous if there exists a single incoming edge into each vertex of this path before v_{i+1} and a single outgoing edge from each vertex after v_i. Each unique contig is formed by a single unique edge and possibly multiple repeat edges, while repetitive contigs consist of the repetitive edges which were not covered by the unique contigs. The visualization was generated using the graphviz tool (http://graphviz.org).



Supplementary Figure 3

The assembly graph of the WORM dataset.

Edges that were classified as repetitive by Flye are shown in color, while unique edges are black. Flye assembled the WORM dataset into a graph with 127 unique and 61 repeat edges and generated 127 contigs as unambiguous paths in the assembly graph. The visualization was generated using the graphviz tool (http://graphviz.org).



Supplementary Figure 4

Dot plots showing the alignment of reads against the Flye assembly, the Miniasm assembly and the reference *C. elegans* genome.

(a) The reference genome contains a tandem repeat of length 1.9 kb (10 copies) on chromosome X with the repeated unit having length ≈190 nucleotides. In contrast, the Flye and Miniasm assemblies of this region suggest a tandem repeat of length 5.5 kb (27 copies) and 2.8 kb (13 copies), respectively. 15 reads that span over the tandem repeat support the Flye assembly (the mean length between the flanking unique sequence matches the repeat length reconstructed by Flye) and suggests that the Flye length estimate is more accurate. (b) The reference genome contains a tandem repeat of length 2 kb on chromosome 1. In contrast, the Flye and Miniasm assemblies of this region suggest a tandem repeat of length 10 kb and 5.6 kb, respectively. A single read that spans over the tandem repeat supports the Flye assembly. Since the mean read length in the WORM dataset is 11 kb, it is expected to have a single read spanning a given 10.0 kb region but many more reads spanning any 5.6 kb region (as implied by the Miniasm assembly) or 2.0 kb region (as implied by the reference genome). Six out of 23 reads cross the "left" border (two out of 18 reads cross the "right" border) of this tandem repeat by more than 5.6 kb, thus contradicting the length estimate given by Miniasm and suggesting that the Flye length estimate is more accurate. (c) The reference genome contains a tandem repeat of length 3 kb on chromosome X. In contrast, the Flye and Miniasm assemblies of this region suggest a tandem repeat of lengths 13.6 kb and 8 kb, respectively. A single read that spans over the tandem repeat reveals the repeat cluster to be of length 12.2k, which suggests that the Flye length estimate is more accurate. (d) The reference genome contains a tandem repeat of length 1.5 kb on chromosome 1. In contrast, the Flye and Miniasm assemblies of this region suggest tandem repeats of length 17 kb and 4.3 kb, respectively. One read that spans over the tandem repeat reveals the repeat cluster to be of length 18.0 kb, which suggests that the Flye length estimate is more accurate.

Supplementary Information for the "Assembly of Long Error-Prone Reads Using Repeat Graphs" manuscript

SUPPLEMENTARY NOTES

Supplementary Note 1: Additional information on benchmarking

Running QUAST. QUAST 5.0 was run using the '--large' option for all eukaryotic genomes, which is recommended for the analysis of large genomes with complex repeat structures. The *minimum alignment identity* was set to a low 90% to account for the higher error rate in some regions of SMS assemblies. The minimum contig length was set to 50 kb for the HUMAN/HUMAN+ assemblies and 5 kb for all of the other assemblies.

Software versions used. All assemblies were run with the default parameters. The exact command lines (and the Falcon configuration script) can be found in the supplementary data archive.

- 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)

The HUMAN (but not the HUMAN+) assembly was generated with the earlier Flye version 2.3.2 (released on Feb 20 2018) to provide a fair comparison with the Canu and MaSuRCA assemblies (which were not updated since the release of Flye 2.3.2). We note that the HUMAN assembly using the latest Flye version 2.3.5 has NGA50 = 7.3 Mb and improves over the Flye 2.3.2 assembly (NGA50 = 6.3Mb). HUMAN+ was assembled using the latest Flye and Canu versions (as of September 2018).

Information about running time and memory usage. Table S1 provides information on the running time and memory usage of various SMS assemblers for the YEAST and WORM datasets

Flye took ≈5,000 CPU hours to generate assemblies of the HUMAN+ dataset using an Intel(R) Xeon(R) 8164 CPU @ 2.00GHz. RAM usage was 500GB at peak. The Canu authors reported ≈30,000 CPU hours of run-time using a cluster with 48-core Intel(R) Xeon(R) CPU @ 2.5GHz with 128 Gb of RAM each (24 nodes) and two 80-core 1 Tb machines. The memory usage of a single job did not exceed 120GB. The MaSuRCA authors reported needing approximately 50,000 CPU hours.

Dataset	Assembler	Wall clock time	Peak memory usage
YEAST-PB 12 Mb genome, 31x 8 threads max	Flye (w/o polishing)	20m (9m)	7G
	Canu	80m	5G
	Falcon	62m	10G
	HINGE	9m	5G
	Miniasm+ABruijn (Miniasm)	16m (1m)	5G
YEAST-ONT 12 Mb genome, 31x 8 threads max	Flye (w/o polishing)	19m (12m)	7G
	Canu	184m	6G
	Falcon	103m	11G
	HINGE	11m	8G
	Miniasm+ABruijn (Miniasm)	31m (3m)	5G
WORM 100 Mb genome, 40x 24 threads max	Flye (w/o polishing)	128 m (77 m)	30G
	Canu	780 m	41G
	Falcon	945m	18G
	HINGE	803m	52G
	Miniasm+ABruijn (Miniasm)	290m (10m)	23G

Table S1. Running time and memory usage of various SMS assemblers. We used a desktop machine with an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (up to 8 threads available) for the YEAST dataset assemblies and a single computational node with an Intel(R) Xeon(R) CPU X5680 @ 3.33GHz for the WORM dataset assemblies (up to 24 threads available). Since we performed the ABruijn polishing step on the Miniasm output, the running time for Flye and Miniasm are given for runs with and without contig polishing; e.g., 25m (9m) for Flye in the case of YEAST-PB dataset indicates 9 m without polishing and 25 m with polishing.

Supplementary Note 2: Assemblies of the METAGENOME dataset

Table S2 presents information about the Flye and Canu assemblies of the METAGENOME dataset.

			Flye			Canu		
bacteria	length (kb)	coverage	% assembled	NGA50 (kb)	#mis-assemblies	% assembled	NGA50 (kb)	#mis-assemblies
A. baumannii	3,976	40	99.8%	906	18	99.8%	906	19
A. odontolyticus	2,393	41	99.5%	622	6	99.8%	**1,285**	5
B. cereus	5,224	25	99.8%	**2,716**	4	99.5%	581	4
B. vulgatus	5,163	46	99.6	**832**	18	98.9%	539	20
D. radiodurans	3,060	52	99.5%	253	25	99.6%	224	27
E. faecalis	2,793	43	99.9%	2,738	0	99.9%	2,747	0
E. coli	4,640	46	99.9%	4,637	0	99.9%	4,643	0
H. pylori	1,667	317	100%	165	2	100%	**1,314**	3
L. gasseri	1,894	83	97.9%	898	1	97.7%	969	1
L. monocytogenes	2,944	98	96.4%	**2,008**	0	100%	1,507	1
P. acnes	2,560	65	100%	2,560	0	100%	2,566	0
P. aeruginosa	6,264	55	99.9%	4,001	3	99.9%	3,998	9
R. sphaeroides	4,131	24	99.4%	**2,006**	1	90.1%	54	0
S. aureus	2,872	66	98.2%	1,003	0	100%	**1,543**	2
S. epidermidis	2,499	59	99.7%	1,276	1	100%	**2,465**	2
S. agalactiae	2,160	42	98.8%	1,836	0	99.9%	2,159	0
S. mutans	2,032	82	99.9%	**1,554**	0	99.9%	1,085	3

Table S2. Information about of the Flye and Canu assemblies of the METAGENOME dataset. Statistics were computed using MetaQUAST v5.0 with default parameters for the bacterial genomes. Entries in bold highlight five assemblies where Flye significantly improved on Canu and four assemblies where Canu significantly improved on Flye. Flye and Canu produced 84 and 99 misassemblies in total, respectively.

Synthetic metagenomic datasets often contain genomes with inaccurate references that present problems for follow-up benchmarking efforts (Nurk et al., 2017). To estimate the expected number of misassemblies caused by the differences between the assembled and reference bacterial strains, we performed the assembly of each of 17 bacteria separately (*separate* assemblies) by binning the initial reads using alignments to the references and running Flye and Canu on the resulting set of reads (see Table S3). Six out of 17 separate assemblies (*R. sphaeroides, A. baumannii, B. cereus* and *B. vulgatus*) were fragmented into 2-4 contigs per chromosome (by both Flye and Canu), while the remaining 11 resulted in a single contig per chromosome. Nevertheless, metaQUAST reported 92 misassemblies in total for the Flye separate assemblies (and 103 misassemblies for Canu). The misassemblies reported for Flye and Canu were highly correlated: 80% of Flye (70% of Canu) misassembly breakpoints had a matching breakpoint in the Canu (Flye) contigs (two breakpoints are matching if their reference coordinates are within 1 kb; note that a single misassembly might have two breakpoints). We thus concluded that the misasemblies reported by metaQUAST were mainly caused by the differences between the genomes in the METAGENOME sample and the reference genomes rather than assembly artifacts.

			Flye			Canu		
bacteria	length (kb)	coverage	% assembled	NGA50 (kb)	#mis-assemblies	% assembled	NGA50 (kb)	#mis-assemblies
*A. baumannii**	3,976	40	99.8%	906	21	99.8%	906	18
A. odontolyticus	2,393	41	99.8%	1,286	4	99.8%	1,285	5
*B. cereus**	5,224	25	99.6%	4,948	3	99.8%	4,625	3
*B. vulgatus**	5,163	46	99.3%	832	21	99.2%	**1,112**	28
*D. radiodurans**	3,060	52	99.6%	253	31	99.5%	222	31
E. faecalis+	2,793	43	99.9%	2,738	0	99.9%	2,745	0
E. coli+	4,640	46	99.9%	4,638	0	99.9%	4,643	0
H. pylori	1,667	317	100%	1,123	2	100%	**1,617**	2
L. gasseri	1,894	83	97.9%	**1,729**	1	97.8%	961	4
L. monocytogenes+	2,944	98	100%	**2,944**	0	100%	2,151	1
P. acnes+	2,560	65	100%	2,560	0	100%	2,566	0
*P. aeruginosa**	6,264	55	99.8%	1,982	2	99.9%	3,998	6
*R. sphaeroides**	4,131	24	99.9%	2,669	2	99.9%	2,578	0
S. aureus	2,872	66	99.8%	**2,665**	1	100%	1,571	2
S. epidermidis	2,499	59	100%	**2,498**	1	100%	1,319	2
S. agalactiae+	2,160	42	99.7%	**2,155**	0	99.9%	1,602	1
S. mutans+	2,032	82	99.9%	**2,032**	0	99.9%	1,546	1

Table S3. Analysis of the separate assemblies of 17 genomes from the METAGENOME dataset. Initial reads were binned into 17 groups using alignments to their respective references. Flye and Canu produced 92 and 104 misassemblies in total, respectively. Statistics were computed using MetaQUAST v5.0. All genomes but the six marked with "*" (*R. sphaeroides, A. baumannii, B. cereus,* B. *vulgatus, D. radiodurans* and *P. aeruginosa)* were assembled into a single contig per chromosome. Six of the remaining 11 Flye assemblies (marked with "+") had no misassemblies compared to the reference. Canu generated four assemblies without reported errors.

Supplementary Note 3: Human segmental duplications identified by Flye

After all unique edges are removed from the assembly graph of the HUMAN+ dataset, it breaks into connected components formed by repeat edges and reveals putative SDs (which might also include short edges corresponding to unresolved common repeats). Figure S2 shows the distribution of lengths of repeat edges exceeding 5 kb and the distributions of lengths of ultra-long SDs (longer than 50 kb).

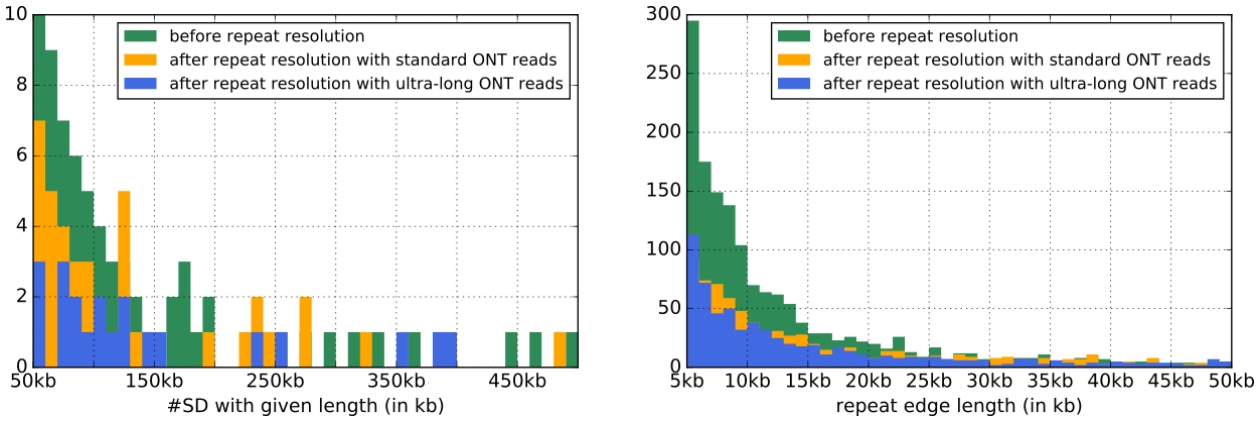


Figure S2. The distribution of lengths of ultra-long SDs (longer than 50 kb) for the assembly graph constructed for the HUMAN+ dataset (left) and the lengths of all other repeat edges (right). (Left) 39 out of 81 SDs (48%) longer than 50 kb were resolved using standard ONT reads. Ultra-long reads resolved an additional 20 SDs (28%) in this range of SD lengths. (Right) Only edges varying in length from 5 kb to 50 kb contributed to the histogram. In addition to these edges, there are 213 (90) repeat edges with length exceeding 50 kb before repeat resolution (after repeat resolution with ultra-long reads). Note that while a similar figure in the main text describes the lengths of SDs (connected components formed by repeat edges), this figure describes the length of individual repeat edges.

We illustrate how Flye resolves unbridged repeats using all five unbridged repeats of multiplicity two in the assembly graph of the HUMAN+ dataset constructed by Flye (Table S4). Flye resolved all five repeats, which range in length from 37 kb to 152 kb, in coverage from 26x to 31x, and in divergence from 1.77% to 7.76%.

All resolved repeats correspond to known segmental duplications in the human genome. The sequences of the constructed repeat copies preferentially mapped to specific copies of segmental duplications, showing that our method is successful even in the presence of Single Nucleotide Polymorphisms (SNVs). For example, repeat 902 aligns to two ≈50 kb regions of chromosome X (separated by ≈65 kb), which are annotated as segmental duplications.

The diploid nature of the human genome may add some complications to the repeat resolution procedure, especially if many SNVs are present in the repeat. However, if the divergence of the repeat significantly exceeds the fraction of SNVs, the described algorithm will still be able to resolve the unbridged repeat. Since the divergence of repeats analyzed in Table S4 (above 4%) significantly exceeds the fraction of SNVs in the human genome (0.1%), SNVs do not significantly affect our approach. However, in the case of unbridged repeats with low divergence (e.g., below 1%), our algorithm has to be modified to take SNVs into account. When the algorithm is extended to repeats of higher multiplicity, it will automatically resolve haplotypes for diploid and polyploid genomes since they will simply be treated as additional repeat copies.

repeat ID	repeat length (kb)	coverage	divergence	#tentative divergent positions	#confirmed divergent positions	maximal distance between divergent positions (kb)	remaining gap (kb)	*#cis* linking reads	*#trans* linking reads
625	152	27x	5.36%	29713	3256	79.2	32.2	2	12
902	51	28x	1.77%	5694	1541	0.7	0	43	13
1018	86	26x	6.77%	17509	11360	0.7	0	17	154
1075	37	28x	3.05%	4379	1406	0.3	0	38	136
1233	49	31x	7.76%	11786	8590	0.3	0	45	2

Table S4. Resolving unbridged repeats of multiplicity two in the assembly graph of the HUMAN+ dataset. The assembly graph of the HUMAN+ dataset has five unbridged repeats of multiplicity two. The identifier of each unbridged repeat is given by its edge identifier in the assembly graph. All repeats have been resolved. The "coverage" is calculated as the total length of reads covering the repeat divided by the repeat length, divided by the multiplicity of the repeat. The "divergence" is calculated based on the alignment of constructed repeat consensus sequences, dividing the total number of substitutions and indels by the total number of matches, substitutions, and indels (if the forward and reverse consensus sequences do not overlap, then the mean divergence of the forward and reverse sequences is calculated, weighted by the length of the sequences). "Maximal distance between divergent positions" refers to the maximum of all distances between adjacent confirmed divergent positions. "Remaining gap" refers to the length of the repeat remaining without separate consensus sequences for each copy after Flye has "moved into the repeat" from both the forward and reverse directions. In the case that the forward and reverse consensus sequences overlap, the remaining gap is set to 0.

Supplementary Note 4: Benchmarking Flye on a simple simulated genome.

We simulated the "genome" shown in Figure 1 with two 99% identical copies of repeat R_1 of length 10 kb and two 99% identical copies of repeat R_2 of length 30 kb. The unique segments A, B, C, and D were simulated as random strings of length ≈250 kb each so that the total genome length is 1 Mb. Afterwards, we simulated reads of length N randomly sampled from this genome at coverage 100x using the PBSIM tool (Ono et al., 2013) and assembled them with Flye. We simulated two sets of reads, one with N = 12 kb (slightly larger than the length of the repeat R_1 but shorter than the length of the repeat R_2) and another with N = 10 kb.

In the case of N = 12 kb, Flye constructed the repeat graph (Figure 1f), identified the bridged repeat R_1, and resolved it as shown in Figure 1h. Afterwards, it resolved the unbridged repeat R_2 and reconstructed its two 99% identical copies (Fig 1i), assembling the entire genome into a single circular contig.

In the case N=10 kb, Flye constructed the repeat graph (Figure 1f), identified both R_1 and R_2 as unbridged repeats and resolved them as shown in Fig 1i. As the result, it assembled the entire genome into a single circular contig.

Supplementary Note 5: Inconsistent pairwise alignments

Pevzner et al., 2004 introduced the concept of alignment-based de Bruijn graphs (*A-Bruijn graphs*) and applied them for repeat characterization and genome assembly. They further described the transformation of an A-Bruijn graph into a repeat graph that is particularly simple in the case of consistent alignments as described below.

Each multiple alignment of m sequences induces $\binom{m}{2}$ pairwise alignments. A set of pairwise alignments (described by the repeat plot) is *consistent* if its alignments can be combined into a single

multiple alignment that induces each pairwise alignment in the set. The concept of multiple alignment is usually defined for the case of aligning multiple sequences rather than for aligning a sequence against itself. Below we describe the concept of a multiple self-alignment of a genome and define the notion of consistent pairwise self-alignments. This notion is important since A-Bruijn graphs result in a simple repeat graph in the case of consistent self-alignments but in a more complex graph in the case of inconsistent self-alignments (see Pevzner et al., 2004 for a discussion of complications arising from inconsistent self-alignments).

A *multiple self-alignment* of a single sequence is a partition of its positions into non-overlapping subsets, with each subset corresponding to a column of the multiple self-alignment. For example, a multiple self-alignment of the sequence ACTGGCTGACT can be represented as a partition of its 11 positions into six "painted" subsets: $A_0C_1T_2G_3G_4C_5T_6G_7A_8C_9T_{10}$. Figure S3 visualizes such a partitioning as a multiple self-alignment where each column represents positions from the same subset:

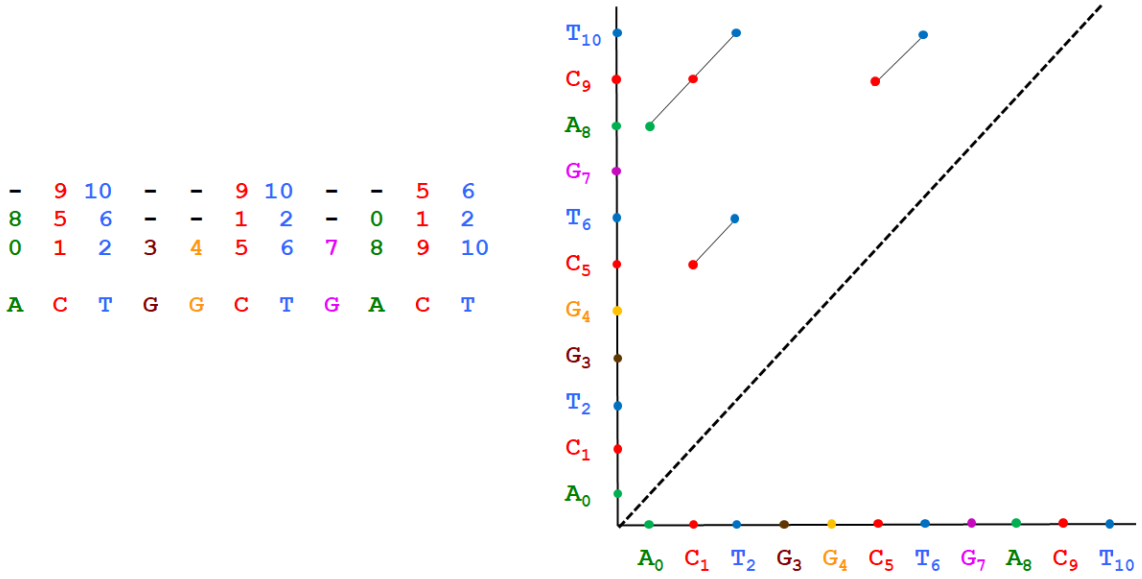


Figure S3. Multiple self-alignment defined by the partitioning of $A_0C_1T_2G_3G_4C_5T_6G_7A_8C_9T_{10}$ into six subsets (left) and the corresponding dot-plot (right). In difference from the traditional representation of a multiple alignment (where each entry represents a nucleotide or a dash in the multiple alignment matrix), each entry in the multiple self-alignment matrix represents a position in the sequence or a dash.

Every pair of numbers $I < j$ in the same column of the multiple self-alignment defines a point (i, j) in the two-dimensional plot. For example, the leftmost column in Figure A5 corresponds to a point (0, 8) and the rightmost column corresponds to points (2, 6), (2, 10), and (6, 10). The collection of all such points defines the *dot plot* of the multiple self-alignment. We refer to a rectangle in the dot plot with lower left corner (x, y) and upper right corner (x', y') as (x, y, x', y'). A pairwise alignment between segments (x, x') and (y, y') of a genome defines a set of two-dimensional points in the rectangle (x, y, x', y') corresponding to matches in this alignment. A multiple self-alignment and a pairwise alignment between segments (x, x') and (y, y') are *consistent* if the dot plot of the multiple self-alignment coincides with the dot plot of the pairwise alignment within the rectangle (x, y, x', y'). A set of pairwise alignments is *consistent* if there exists a multiple self-alignment that is consistent with all pairwise alignments in this set, and *inconsistent* otherwise.

Supplementary Note 6: Inconsistent alignments result in excessively complex repeat graphs

Figure S4 presents an example of inconsistent pairwise alignments and illustrates that they result in a repeat graph that differs from the repeat graph shown in Figure 2 of the main text. In difference from the graph in Figure 2, the graph in Figure S4 is not alignment-compatible; e.g., the repeat A+B corresponds to a single path in Figure 2 but two paths in Figure S4. Although it may appear to be a minor annoyance in the case of the toy example in Figure S4, inconsistent alignments may result in excessively complex repeat graphs in the case of real genomes, making it difficult to analyze repeats in the genome. While it is easy to make the pairwise alignments consistent in the simple case shown in Figure S4 (by adding a missing diagonal), transforming inconsistent pairwise alignments into consistent ones is a challenging task in the case of real genomes.

The approximate repeat graph in Figure S4 has seven vertices (since there exist seven projections of alignment endpoints to the main diagonal), in contrast to the approximate repeat graph in Figure 2 of the main text that has eight vertices. This deficiency of the approximate repeat graph in Figure S4 motivates the algorithm for extending the set *Breakpoints* that is described in the main text. Note that the middle point of the long diagonal in Figure S4 represents an invalid point since only one of its projections (shown as a purple point) belongs to the set of seven endpoint projections on the main diagonal. The algorithm described in the main text adds the missing projection to the set *Breakpoints* and results in the same approximate repeat graph as shown in Figure 2 in the main text (Figure S4, bottom panel).

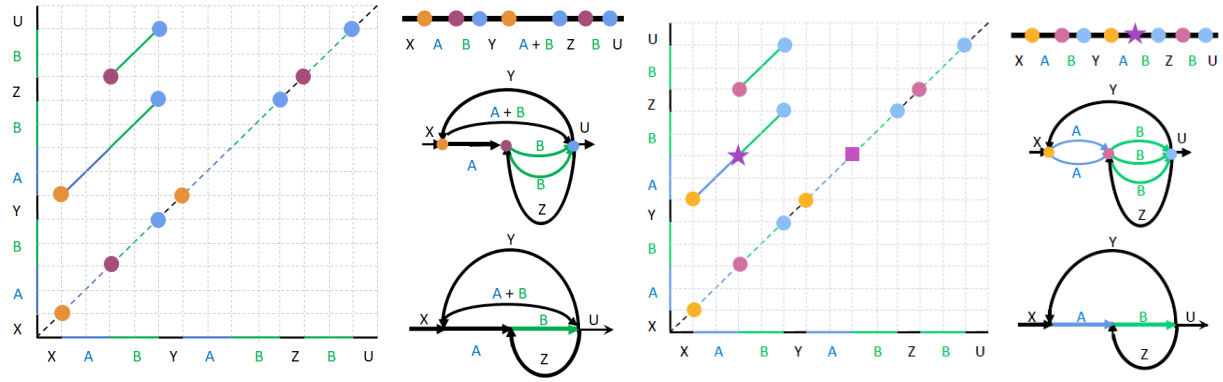


Figure S4. Inconsistent pairwise alignments result in an "incorrect" repeat graph (as compared to the graph shown in Figure 2 in the main text), thus necessitating an extension of the set of alignment endpoints. (Left) Alignment-paths for two pairwise self-alignments within a genome XABYABZBU. Only two out of three pairwise alignments between instances of a mosaic repeat (AB, AB, and B) are shown since the third alignment did not pass the percent identity threshold, resulting in an inconsistent set of pairwise alignments. 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 $d = 0$ are painted with the same color). This clustering reveals three clusters with seven breakpoints on the main diagonal. (Top right) Projections of the clustered endpoints on the main diagonal define seven vertices of the approximate repeat graph. (Middle right). Gluing breakpoints that belong to the same clusters. (Bottom right) Gluing parallel edges in the resulting graph (parallel edges are glued if there exists an alignment between their sequences), which results in an approximate repeat graph that is not alignment-compatible. **(Right)** Extending the set *Breakpoints* by adding an additional point to the longest diagonal (shown as a star). Since the middle point of the longer alignment-path is invalid (its vertical projection on the main diagonal belongs to the set *Breakpoints* but its horizontal projection does not), we have added the missing projection to the set *Breakpoints* (shown as a purple square). Adding this breakpoint is equivalent to breaking the longer alignment-path into two subpaths (the breakage position is shown as a purple star). As a result of the breakpoint extension procedure, the approximate repeat graph constructed based on the extended set *Breakpoints* coincides with the approximate repeat graph shown in Figure 2 of the main text.

Supplementary Note 7: The challenge of assembling contigs into a repeat graph

The ABruijn algorithm constructs a set of contigs but does not attempt to assemble them into even longer contigs (e.g. by utilizing ultra-long reads) and stops short of constructing the repeat graph of the genome (Lin et al., 2016). We note that contig assembly (let alone constructing the repeat graph based on contigs) is a non-trivial problem. Although it may appear that contig assembly can be achieved by simply utilizing existing long read assemblers, Bankevich and Pevzner, 2015 reported that Celera (Myers et al., 2000), Minimus (Treangen et al., 2013), and Lola (Sharon et al., 2015) assemblers produced suboptimal assemblies of contigs generated using *TruSeq Synthetic Long Reads (TSLR)* technology. Their attempts to modify the short read assembler SPAdes (Bankevich et al., 2012) for TSLR assembly improved on the results of Celera, Minimus, and Lola but stopped short of constructing the contig-based repeat graph.

Similar challenges remain unresolved in the case of short reads. Although popular short read assemblers construct the assembly graph of *single* reads (before resolving repeats using paired reads), they output a set of contigs (after the repeat resolution step) rather than an assembly graph that utilizes information about *paired* reads. For example, SPAdes (Bankevich et al., 2012) constructs the assembly graph of single reads, uses it together with paired reads for repeat resolution, and outputs the resulting contigs (Prjibelsky et al., 2014). A better option would be to construct the assembly graph of these contigs (which is less tangled than the assembly graph of individual reads) and to apply the repeat resolution step again to this graph. Another advantage of this (less tangled) contig-based assembly graph lies in applications relating to hybrid assembly, e.g., co-assembly of short and long reads (Antipov et al., 2015, Wick et al., 2017). However, although some studies attempted to construct the assembly graph from contigs or directly from paired reads (Vyahhi et al., 2012), the popular short read assemblers have failed to incorporate this approach into their pipelines so far.

Supplementary Note 8: `FlyeWalk` algorithm

The **`FlyeWalk`** algorithm (shown below) computes alignments (within the **Overlap**, **MapReads,** and **ExtendRead** procedures) using the longest jump-subpath approach described in Lin et al., 2016. In difference from other SMS assemblers, `FlyeWalk` does not generate all-versus-all pairwise alignments between reads (a major time bottleneck) since reads that align to a newly assembled disjointig are removed from the set *UnprocessedReads*.

```
FlyeWalk(AllReads, MinOverlap)
    Disjointigs ← empty set of contigs
    UnprocessedReads ← AllReads
    for each Read in UnprocessedReads
        ChainOfReads ← ExtendRead(UnprocessedReads, Read, MinOverlap)
        DisjointigReads ← MapReads(AllReads, ChainOfReads, MinOverlap)
        DisjointigSequence ← Consensus(AllReads, DisjointigReads, MinOverlap)
        add DisjointigSequence to Disjointigs
        remove DisjointigReads from UnprocessedReads
    return Disjointigs

ExtendRead(UnprocessedReads, Read, MinOverlap)
    ChainOfReads ← sequence of reads consisting of a single read Read
    while forever
        NextRead ← FindNextRead(UnprocessedReads, Read, MinOverlap)
        if NextRead = empty string
          return ChainOfReads
        else
          add NextRead to ChainOfReads
          Read ← NextRead
          remove Readfrom UnprocessedReads
```

Pseudocode of the `FlyeWalk` algorithm. `FlyeWalk` iteratively extends each unprocessed read and organizes the selected reads into a chain. Each such chain contributes to a disjointig and **`FlyeWalk`** outputs the set of all disjointigs resulting from such extensions. **ExtendRead** generates a random walk in the assembly graph, which starts at a given read and constructs a chain of overlapping reads that contribute to a constructed disjointig. It terminates when there are no unprocessed reads overlapping the current read by at least *MinOverlap* nucleotides. **FindNextRead** finds an unprocessed read that overlaps with the given read by at least *MinOverlap* nucleotides and returns an empty string if there are no such reads. **MapReads** finds all reads that align to a given chain of reads over their entire length with the possible exception of a short suffix and/or prefix of length at most *MinOverlap*. **Consensus** constructs the consensus of all reads that contribute to a given disjointig.

Given a chain of reads *ChainOfReads* formed by reads $Read_1 \ldots Read_n$, we define $prefix(Read_i)$ as the overlapping region between consecutive reads $Read_{i-1}$ and $Read_i$ in the chain and $suffix(Read_i)$ as the suffix of the *i*-th read after the removal of $prefix(Read_i)$ (note that $suffix(Read_1)$ coincides with $Read_1$). We define $concatenate(ChainOfReads)$ as the concatenate $suffix(Read_1)*\ldots*suffix(Read_n)$ of read suffixes in this chain. The **Consensus** procedure constructs an initial draft sequence (disjointig) of a chain *ChainOfReads* by constructing $concatenate(ChainOfReads)$. Afterwards, all reads from the dataset are aligned to the draft disjointig sequence using minimap2 (Li, 2017) and the consensus of the aligned reads is formed by taking the majority vote. This procedure reduces the error rate in the draft disjointig sequence from ≈13% to 1-5%, depending on the contig coverage. The follow-up polishing step reduces the error rate to ≈0.1% when the coverage exceeds 30x.

ExtendRead is run in a single thread but multiple **ExtendRead** procedures are run in parallel for each read that is not in *UnprocessedReads.* When one of the **ExtendRead** procedures finishes, the algorithm checks if the returned disjointig has a significant overlap (by more than 10% of its length) with one of the previously constructed disjointigs from *Disjointigs*. If such an overlap is found, the new disjointig is

discarded and the reads from this disjointig are returned to the set *UnprocessedReads*. This parallelization significantly speeds up `FlyeWalk` for assemblies that contain many contigs.

Supplementary Note 9: Flye constructs an accurate assembly graph from error-prone disjointigs

There exist two tours in the assembly graph for the *E. coli* strain NCTC9964 shown in Figure 3 (middle) of the main text: the correct genomic tour formed by edges IN_1, *REP*, OUT_1, and *REP'* (the corresponding complementary tour is formed by the complementary edges *REP*, OUT_2, *REP'*, and IN_2) and the incorrect tour formed by edges IN_1, *REP*, OUT_2, and *REP'* (the corresponding complementary tour is formed by edges IN_2, *REP*, OUT_1, and *REP'*).

Although paths $IN_1 \rightarrow REP \rightarrow OUT_2 \rightarrow REP'$ and $IN_2 \rightarrow REP \rightarrow OUT_1 \rightarrow REP'$ form incorrect disjointigs, they are however assembled in the correct assembly graph by Flye. Below we explain why an arbitrary set of paths (disjointigs) constructed by FlyeWalk results in a correct assembly graph. Although our arguments apply to the punctilious repeat graph, the construction of the approximate repeat graph follows a similar logic, and the Results section demonstrates that these graphs constructed by Flye also result in accurate assemblies.

Let *Genome* be an (unknown) genomic sequence of an (unknown) length with an (unknown) alignment matrix *Alignments*. Let *Strings* = {*s(1), ..., s(t)*} be a covering set of strings for *Genome*, and *A(i, j)* be the alignment snapshot, i.e., the sub-matrix of *Alignments* for substrings *s(i)* and *s(j)*. Given a concatenate $Strings^* = s(1) * s(2) * \ldots * s(t)$ of all t substrings (with delimiters), their $t * (t - 1) / 2$ pairwise alignment snapshots *A(i, j)* can be combined together to form the alignment matrix *Alignment** of the entire concatenate. We emphasize that the coordinates of the strings *s(1), ..., s(t)* and their ordering in the sequence *Genome* are unknown.

Pevzner et al., 2004 demonstrated that *RepeatGraph(Genome, Alignments)* coincides with the repeat graph *RepeatGraph(Strings*, Alignments*)* of a concatenate of all substrings (in any order) for any covering set of substrings. As we explain below, this result implies that the Flye assembly of inaccurate disjointigs generated by `FlyeWalk` results in an accurate assembly graph. For simplicity, we assume that chimeric reads have been removed and that no read is contained within another read.

Consider the set of disjointigs $\{disjointig_1, disjointig_2, \ldots, disjointig_t\}$ constructed by `FlyeWalk` and map all reads to all these disjointigs. Since `FlyeWalk` utilizes all reads, each read will be mapped to one or more disjointigs. We now concatenate all reads starting from reads in the first disjointig, followed by reads in the second disjointig, etc., resulting in the sequence of reads:

$$\{s(1, 1), s(1, 2), \ldots, s(1, n_1)\}, \{s(1, 1), s(2, 1), \ldots, s(2, n_2)\}, \ldots, \{s(t, 1), s(t, 1), \ldots, s(t, n_t)\}$$

where *s(i, j)* stands for the *j*-th read in the *i*-th disjointig (reads are ordered in the increasing order of their starting positions in each disjointig).

Since all reads are included in this concatenate, the repeat graph constructed from this concatenate coincides with the repeat graph of the genome (Pevzner et al., 2004). Since the repeat graph does not depend on the order in which the reads are glued, we will perform gluing in two stages. At the first stage, we will perform some (but not all) gluing operations on reads from the first disjointig, followed by some gluing operations on reads from the second disjointig, etc. Specifically, with respect to the *i*-th disjointig, we will only glue overlapping reads within this disjointig (i.e., reads *s(i, n)* and *s(i, m)* if $n < m$ and read *s(i, m)* starts before read *s(i, n)* ends) and will only apply gluing operations to their overlap. Note that the

first gluing stage does not necessarily includes all gluing operations applicable to reads from the *i*-th disjointig, e.g., non-overlapping reads within this disjointig may share sufficiently long alignments that however do not contribute to the first-stage gluing.

The first-stage gluing of reads that were sampled from a single disjointig results in the consensus of this disjointig constructed by `FlyeWalk`. Thus, the application of such "intra-disjointig" gluing operations to all disjointigs results in the set of disjointigs $\{disjointig_1, disjointig_2, ..., disjointig_t\}$. Note that only some but not all gluing operations have been performed at this point; e.g., inter-disjointig gluing has not been applied yet. Therefore, the second-stage gluing of all disjointigs constructed by `FlyeWalk` (some of them may be misassembled) results in the same assembly graph as gluing all reads, and thus results in the repeat graph of the genome.

Supplementary Note 10: Constructing the repeat graph from substrings of a genome

The repeat graph construction algorithm assumes that the genome *Genome* and the two-dimensional matrix *Alignments (*defining the pairwise alignments between similar substrings of the genome) are given. Any two substrings of the genome define a rectangle in the matrix *Alignments* that we refer to as an *alignment snapshot* imposed by these substrings. Given a set of substrings from *Genome*, Pevzner et al., 2004 asked whether the repeat graph can be constructed from their pairwise snapshots without knowing *Genome* and the entire matrix *Alignments.* This question is relevant to genome assembly when the *Genome* and *Alignments* are unknown but the alignments between substrings of the genome (reads) can be computed as an approximation of alignment snapshots.

A set of substrings of a genome forms a *covering set* if, for every pair of consecutive positions in *Genome*, there exists a substring containing these positions. Pevzner et al., 2004 demonstrated that if substrings of a genome (reads) form a covering set, then gluing an arbitrary concatenate of these substrings (separated by delimiters), according to their alignment snapshots, produces the same repeat graph as gluing the entire *Genome*.

This result holds for the ideal case when the alignment snapshots are inherited from the matrix *Alignments* representing all self-alignments of *Genome.* Since *Genome* and the matrix *Alignments* are unknown in the case of genome assembly, the alignment snapshot between two substrings (reads) is computed as their pairwise alignment rather than derived as the corresponding rectangle in the *Alignments* matrix. This pairwise alignment may differ from the alignment snapshot; for example, an alignment between two reads overlapping by a single nucleotide will be captured in their alignment snapshot (since it is a part of a larger matrix *Alignments*) but not in their pairwise alignment since it does not pass a statistical significance threshold. That is why Pevzner et al., 2004 introduced a more stringent condition for the concept of the covering set of substrings: for each *m* consecutive positions in *Genome* (where *m* is a pre-defined threshold), there must exist a substring (read) spanning all these positions. This condition explains why it is important that Flye generates disjointigs satisfying the overhang property.

Supplementary Note 11: Aligning reads to the assembly graph

Flye aligns all reads to the constructed assembly graph using the concept of *common jump-subpaths* (Lin et al., 2016). First, each read is matched against the edges of the assembly graph. For each repeat edge in the assembly graph, we store all copies of the corresponding repeat (from the original disjointigs), rather than a single consensus of all sequences contributing to this repeat edge. We then match a read to *all* these copies and select the best alignment to improve the recruitment of reads to the edges of the assembly

graph. If a read is aligned to multiple edges in the assembly graph, we find a maximum scoring path in the graph formed by such edges using dynamic programming.

Supplementary Note 12: Identifying repeat edges in the assembly graph.

After constructing the assembly graph, Flye aligns all reads to this graph and forms a *read-path* for each read. Given the alignments of all reads against the assembly graph, Flye computes the mean depth of coverage *cov* across the entire assembly graph and classifies an edge as *low-coverage* (if its coverage is below 2**cov*) and *high-coverage* (if its coverage is at least 2**cov*). While most low-coverage edges are unique (traversed only once in the genomic tour), some of them are repetitive since the coverage varies along the genome.

To improve the classification of unique and repetitive edges in the assembly graph, Flye reclassifies some edges using information about the read-paths. An edge *e'* in the assembly graph is a *successor* of an edge *e* if it follows *e* in one of the read-paths. A low-coverage edge is classified as *unique* if it has a single successor. All other edges (i.e., high-coverage edges and low-coverage edges with multiple successors) are classified as repetitive.

To avoid classifying chimeric connections in the assembly graph as successor edges and to minimize the influence of misaligned reads, Flye imposes an additional restriction on the edge to classify it as a successor: a fraction of the reads supporting a successor (among all reads contributing to the successor of a given edge) should exceed *N* percent of the fraction of the reads supporting the most frequent successor (the default value is $N = 20\%$).

We used the Flye *C. elegans* assembly to estimate the accuracy of Flye's classification of unique and repetitive edges. For each edge in the *C. elegans* assembly graph, we found whether it is unique or repetitive in the reference genome by aligning the edge to the entire reference genome and checking whether there exists a single alignment (unique edge) or multiple alignments (repetitive edge). This analysis revealed that the *C. elegans* assembly graph has 339 unique and 219 repetitive edges. Flye has misclassified 5 out of 219 repetitive edges as unique (2%) and 22 out of 339 unique edges as repetitive (6%). Note that only errors of the first type (misclassifying repeat edges as unique) lead to potential misassemblies during the repeat resolution step. Errors of the second type (classifying unique edges as repeat edges) do not lead to misassemblies but may potentially negatively affect the contiguity of the assembly since misclassified unique edges do not contribute to repeat resolution. This is however not a critical shortcoming in practice since long reads often bridge these misclassified edges.

Supplementary Note 13: Additional details on untangling assembly graphs

The maximum weight matching defines the set of edges in the transition graph; Flye additionally checks each of the inferred edges as follows. For each edge *(u, v)* from the matching, it computes the total weight *TotalWeight* of all edges in the transition graph adjacent to *u* or *v*. If $transition(u, v) < TotalWeight / 2$, the edge is classified as *weak* and is consequently ignored. Weak edges typically arise from long repeats that may be bridged by a few reads in an ambiguous way.

Flye iteratively untangles edges and finds maximum weight matchings until no extra repeats can be resolved. Note that a repeat of multiplicity *t* may require less than *t* untangling operations to be completely resolved. For example, a repeat edge *REP* of multiplicity 2 in the assembly graph (with

incoming edges IN_1 and IN_2 and outgoing edges OUT_1 and OUT_2) may only have bridging reads traversing IN_1, *REP,* and OUT_1 but not IN_2, *REP*, and OUT_2. However, using bridging reads to untangle IN_1 and OUT_1 (essentially forming a single edge from edges IN_1, *REP,* and OUT_1), turns the sequence of edges IN_2, *REP*, and OUT_2 into a non-branching path and thus completely untangles the repeat.

Note that some short edges are reclassified as long during this process and that some repetitive edges are reclassified as unique during the next iteration of the algorithm (for example, if they were a part of a bigger mosaic repeat that was partially resolved).

Supplementary Note 14: Revealing variable positions within repeats

To reveal the variable positions within a repeat (a repeat edge in the assembly graph), we map all reads to the consensus sequence of the repeat and generate a multiple alignment of all reads that are contained within or overlap with the repeat. Afterwards, we determine the second most frequent nucleotide in each column of the multiple alignment and define the *substitution* rate in this column as the number of occurrences of the second most frequent nucleotide divided by the total number of reads covering this column (note the difference between the concepts of substitution rate here and in Lin et al., 2016). We define the *deletion* and *insertion* rates in each column as in Lin et al., 2016. If the substitution, deletion, or insertion rate for a column exceeds a predefined threshold, the corresponding position is called a *tentative divergent position*. The *repeat divergence* is estimated by dividing the total number of tentative divergent positions by the length of the repeat. See Supplementary Note "Human segmental duplications identified by Flye" for a discussion on how repeat divergence can be affected by diploidy.

Below we construct the distribution of substitution, deletion, and insertion rates in non-divergent positions, compare it with the distribution of substitution, deletion, and insertion rates in divergent positions, and select a threshold that separates these two distributions. To construct these distributions, we selected the 22 kb long repeat (repeat *REP* in Figure 3) in the assembly graph of the EC9964 dataset (other repeats result in very similar distributions). Since this repeat has many variations between two repeat copies, (943 substitutions (4.3%), 346 deletions (1.6%), and 226 insertions (1.0%)), we manually resolved it with high confidence. A position in this repeat is classified as *variable* if it corresponds to a substitution, deletion, or insertion, and *non-variable* otherwise.

We mapped reads from the EC9964 dataset to the consensus of the *REP* repeat and calculated the substitution, deletion, and insertion rates. Figure S5 illustrates that variable positions feature higher substitution, deletion, and insertion rates than non-variable positions within a repeat. We thus identify tentative divergent positions based on mutation rates by selecting a mutation rate threshold that provides a good separation between the two distributions (0.1, 0.2, and 0.3 for substitutions, deletions, and insertions, respectively). This results in the identification of 924 out of 943 substitutions, 270 out of 346 deletions and 54 out of 226 insertions for the *REP* repeat. At the same time, we misclassified 81 non-variable positions as divergent (61 substitutions, 5 deletions, and 15 insertions), resulting in a false positive rate of 0.4%. In all, we identified 1329 tentative divergent positions, which leads to a divergence estimate of 6.0%, a slight underestimation of the true divergence rate of 6.9%

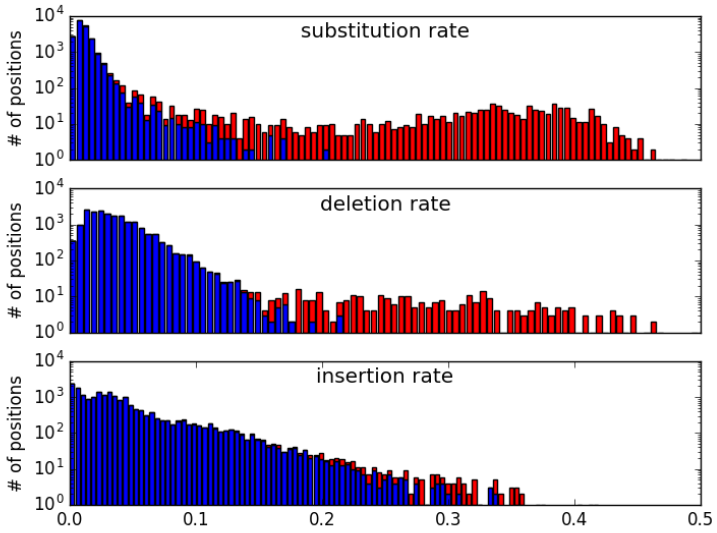


Figure S5. Separating variable and non-variable positions within repeats using substitution, deletion, and insertion rates computed for the *REP* repeat in the EC9964 dataset. Substitution (top), deletion (middle), and insertion (bottom) rates at each position in the multiple alignment of reads. Blue (red) bars represent mutation rates for non-variable (variable) positions. The number of positions with a given mutation rate (*y*-axis) is shown in a logarithmic scale. The cutoffs 0.1, 0.2, and 0.3 result in a good separation between variable and non-variable positions for substitutions, deletions, and insertions, respectively.

Supplementary Note 15: Additional details on the unbridged repeat resolution approach

Initially, the unbridged repeat resolution algorithm recruits all reads traversing edges IN_1 and *REP* (IN_2 and *REP*) to the first (second) repeat copy and computes the consensus of each repeat copy using the recruited reads. Since the recruited reads do not span the entire edge *REP*, we only construct two consensus sequences corresponding to prefixes of *REP* where there is substantial read coverage by the recruited reads. We require at least *minCoverage* for each repeat copy to ensure that consensus sequences are sufficiently accurate (the default value of *minCoverage* = 10). Both consensus sequences are truncated to the length of the shortest consensus sequence to prevent bias in the read recruitment process in future iterations. In the case of the *REP* repeat in the EC9964 dataset, we constructed two consensus sequences corresponding to 8.6 kb long prefixes of *REP* with divergence 9.8% (Figure 3). As a result, we now have two consensus sequences for the entire edge *REP* that differ in some of the first 8.6 kb but coincide in the remaining part. The two constructed consensus sequences serve as two templates for recruiting reads to specific repeat copies in successive iterations. In this way, we gradually construct the consensus sequences from only reads that have been assigned to a specific repeat copy with high confidence.

This brief description hides some details, e.g., it is not clear why we identified the set of putative divergent positions since these positions have not been mentioned in the description of the algorithm. In reality, the constructed consensus sequences of prefixes of two repeat copies may have errors since the read coverage of these prefixes may be as low as the default parameter *minCoverage* = 10x. Indeed, the consensus sequences are expected to have a high 7.5% error rate when the coverage is as low as 5-10x (Lin et al., 2016). Since these error-prone consensus sequences serve as two templates for recruiting reads to specific repeat copies in successive iterations, the read recruitment is compromised. We thus recruit reads to specific repeat copies based only on tentative divergent positions in the repeat. Since these positions were identified based on all reads (full coverage) rather than only reads contributing to a given template (coverage as low as *minCoverage*), they provide a more reliable standard for read recruitment.

Below we provide a description of various steps during the unbridged repeat resolution:

- *Evaluating the tentative divergent positions.* We map all classified reads again, this time to two consensus copies of the repeat (rather than a single consensus copy as in the initial iteration) to construct a more accurate alignment. We further utilize the set of tentative divergent positions that were identified at the initial stage of the algorithm. We consider the consensus sequence of each repeat copy and compare the most frequent symbols (A, C, G, T, –) occurring in the set of already classified reads for each repeat copy at each tentative divergent position. If the most frequent symbol at a position differs for two repeat copies, then that position is called a *confirmed divergent position*. The most frequent symbols of all the confirmed divergent positions for a certain repeat copy represent a "signature" of this copy. Since some positions within a repeat may not have been reached by the two consensus sequences yet, they remain classified as tentative divergent positions.
- *Assigning reads to various repeat copies.* We now map all unclassified reads to two consensus copies of the repeat and utilize the confirmed divergent positions to assign unclassified reads to a specific repeat copy. For each read, we compute its *vote* for each repeat copy as the number of confirmed divergent positions at which the symbol of the read agrees with the consensus of this repeat copy (all other positions are ignored). The read is assigned to a specific repeat copy if its vote for this copy is larger than the vote for another copy by a minimum threshold (the default value is three). The read remains unassigned in the case of ties.
- *Constructing new consensus sequences for each repeat copy.* We use all reads that have been assigned to a specific repeat copy to construct a new consensus sequence for this copy. The consensus is only constructed up to where the coverage of the reads is at least *minCoverage* in both repeat copies to ensure that consensus sequences are accurate, and then both consensus sequences are truncated to the length of the shortest consensus sequence. The algorithm then proceeds to the next iteration unless no new reads mapping to the original repeat consensus were classified or all of the consensus sequences are identical to those in the previous iteration, in which cases it terminates.

Although we discussed the algorithm as "moving forward" into the repeat (e.g., moving ahead from edges IN_1 and IN_2 in Figure 3), the same procedure is performed by "moving backward" in the opposite direction (e.g., moving backwards from edges OUT_1 and OUT_2 in Figure 3), or equivalently, moving forward along the reverse complement of the repeat. There are two stopping rules for the described algorithm: (i) when the prefix of the repeat resulting from "moving forward" overlaps with the suffix of the repeat resulting from "moving backward" and (ii) when the prefix and the suffix both stop extending but still do not overlap. At this point, a consensus sequence has been constructed for both prefix and suffix of each repeat copy and a set of confirmed divergent positions for each repeat copy has been obtained.

As the repeat consensus sequences have been extended forward and backward (Figure 3), this procedure may result in the emergence of *linking reads,* i.e., reads that are assigned to both a repeat copy originating from one of the incoming edges (IN_1 or IN_2) and a repeat copy originating from one of the outgoing edges (OUT_1 or OUT_2). Linking reads are grouped depending on which incoming/outgoing edges they are assigned to: IN_1 and OUT_1, IN_2 and OUT_2, IN_1 and OUT_2, or IN_2 and OUT_1. We further classify all linking reads into one of two categories called *cis* (IN_1/OUT_1 and IN_2/OUT_2) and *trans* (IN_1/OUT_2 and IN_2/OUT_1) since there are only two ways to resolve the repeat: pairing IN_1/OUT_1 with IN_2/OUT_2, or pairing IN_1/OUT_2 with IN_2/OUT_1.

If the number of linking reads in one of the categories exceeds a threshold (the default value is five) and exceeds the number of linking reads in another category by at least a factor of two, all reads in the "winning" category are assigned to the corresponding repeat copies and the consensus of each repeat copy is computed based on all reads assigned to this copy.

If our attempts to resolve the repeat did not result in the emergence of linking reads or if the conditions above on the number of linking reads do not hold, the repeat is classified as unresolved (note that some resolvable repeats may be classified as unresolved). Note that even in the case of unresolved repeats, our algorithm still finds more accurate consensus sequences for the prefixes and suffixes of the repeat. Supplementary Note "Resolving unbridged repeats in the BACTERIA dataset" analyzes resolvable and unresolvable repeats in more detail.

Our analysis of the BACTERIA dataset suggests that a repeat can usually be classified as resolvable based on the following two criteria:

- *The divergence rate exceeds a minimum divergence threshold.* Based on simulated data, we set up a minimum 0.1% divergence threshold, i.e. at least one divergent position per each 1000 nucleotides on average. When the divergence rate falls below 0.1%, there is often a shortage of reads covering multiple divergent positions, which is necessary for successful repeat resolution. To determine the minimum divergence threshold for which the repeat resolution algorithm could be applied successfully, we simulated several repeats of multiplicity two of length 10 kb, 20 kb, and 40 kb, with divergence rates ranging from 0.01% to 0.45%. Variations between the different copies of these repeats were introduced by adding substitutions and indels randomly to both copies until the desired divergence rate was reached. Next, we simulated PB reads from these repeats with coverage 100X, mean error rates of 15%, and read lengths between 5 kb and 15 kb. When the repeat resolution algorithm was applied to these datasets, we found that all simulated repeats with divergence rate greater than 0.1% were successfully resolved. We thus chose 0.1% to be the minimum divergence threshold.
- *The distance between consecutive putative divergent positions does not exceed the maximum distance threshold.* If consecutive divergent positions are 15 kb apart but the maximal read length is 10 kb, there will be no reads spanning these positions that can be used for repeat resolution. Moreover, it turns out that the maximal read length is too optimistic of a threshold, since the repeat may still be unresolvable even if consecutive divergent positions are less than the length of a read away. For example, although there are many divergent positions in a 24 kb long repeat of multiplicity two in the EC9007 dataset, there exists a 8 kb gap between consecutive divergent positions (located at positions 15,002 and 23,150 from the start of the repeat). The repeat is classified as unresolved since there is only one read spanning this gap, which does not provide a confident pairing of the incoming and outgoing edges for this repeat. On the other hand, we found that selecting the average read length as the threshold is too lenient. Based on our analysis of the BACTERIA dataset, we set the default threshold for the maximal distance between consecutive divergent positions as twice the average read length, which varies from 12 kb to 20 kb in the BACTERIA datasets.

If either of the above criteria does not hold, the repeat is classified as unresolvable.

Supplementary Note 16: Evaluating the accuracy of the unbridged repeat resolution approach

To evaluate the accuracy of the unbridged repeat resolution approach, we simulated a 1 Mb genome which contains two copies of a single repeat of length L (for L = 10 kb, 20 kb, and 40 kb) with divergence x% (for x = 0.05%, 0.15%, and 0.45%). We further used the PBSim tool (Ono et al. 2013) to simulate Pacific Biosciences reads from this genome with coverage 100x and length varying from 5 kb to 15 kb. The PBSim tool generated reads with an insertion rate of 9%, a deletion rate of 4.5%, and a substitution rate of 1.5%. We also generated two replicates for each simulation, performing 3*3*2 = 18 simulations in total.

All repeats of length 10 kb were resolved by Flye prior to unbridged repeat resolution, so they were not included in this analysis. Of the remaining repeats, all repeats with divergence 0.15% and 0.45% were resolved correctly with overwhelming support from linking reads, and the repeat copy sequence reconstructions had accuracy >99.95%. All repeats with divergence 0.05% were not resolved due to the scarcity of divergent positions, but over half of the sequences of each repeat copy were reconstructed with accuracy exceeding 99.8%.

We further lowered the divergence rate to 0.01%, 0.02%, 0.03%, and 0.04% and repeated the experiment. With such low divergence rates, the repeat sequences could not be fully reconstructed, but all reconstructed sequences still had ≥99.9% accuracy. We also lowered the coverage to 30X and 50X and repeated the experiment with similar results (Table S5-S7).

Based on these simulations, we conclude that our unbridged repeat resolution approach correctly links incoming and outgoing edges using the evidence from linking reads, and it can successfully reconstruct the sequences of distinct repeat copies. Even at low divergence rates and low coverage, our reconstructions are very accurate though we may only reconstruct partial sequences due to regions with no divergent positions or low coverage.

length (kb)	cov	div (%)	rep	#true divergent positions	#confirmed divergent positions	maximal distance between positions (kb)	#linking reads	resolved	mean reconstructed sequence length (kb)	mean reconstructed sequence accuracy
20	100x	0.05	A	7	7	5.0	0	NO	14	99.93%
			B	9	10	10.0	0	NO	20	99.79%
		0.15	A	32	40	2.8	439	YES	20	99.98%
			B	35	41	2.3	462	YES	20	99.97%
		0.45	A	103	109	0.87	471	YES	20	99.96%
			B	81	88	1.5	552	YES	20	99.98%
40	100x	0.05	A	19	8	28.0	0	NO	40	99.88%
			B	18	6	32.9	0	NO	15	99.94%
		0.15	A	68	82	2.8	840	YES	40	99.95%
			B	64	73	2.9	788	YES	40	99.97%
		0.45	A	194	215	1.0	842	YES	40	99.97%
			B	169	188	1.5	861	YES	40	99.97%

Table S5. Unbridged repeat resolution simulation results. Genomes containing repeats of multiplicity two were simulated for repeat lengths 20 kb and 40 kb, and divergence rates 0.05, 0.15, and 0.45. PB reads were simulated for these genomes with a mean error rate of 15% and lengths ranging from 5 kb to 15 kb. Our unbridged repeat resolution approach was applied to these reads to determine how these repeats should be resolved (by pairing incoming with outgoing edges) and to reconstruct the distinct sequences of repeat copies for each of these simulations.

length (kb)	cov	div (%)	rep	#true divergent positions	#confirmed divergent positions	maximal distance between positions (kb)	#linking reads	resolved	mean reconstructed sequence length (kb)	mean reconstructed sequence accuracy
20	100x	0.01	A	1	1	19.5	0	NO	11	99.96%
			B	2	2	17.7	0	NO	11	99.95%
		0.02	A	2	2	13.0	0	NO	12	99.93%
			B	3	5	10.2	0	NO	16	99.96%
		0.03	A	6	8	7.8	0	NO	13	99.92%
			B	4	5	13.5	0	NO	12	99.94%
		0.04	A	7	7	7.8	0	NO	14	99.96%
			B	7	6	13.4	0	NO	11	99.92%
40	100x	0.01	A	5	4	34.2	0	NO	13	99.94%
			B	3	4	33.3	0	NO	12	99.94%
		0.02	A	7	8	26.3	0	NO	15	99.96%
			B	6	1	38.0	0	NO	15	99.91%
		0.03	A	14	5	28.0	0	NO	16	99.93%
			B	8	2	31.3	0	NO	14	99.94%
		0.04	A*	17	4	29.7	0	NO	13	99.89%

Table S6. Unbridged repeat resolution low divergence simulation results. Genomes containing repeats of multiplicity two were simulated for very low divergence rates from 0.01 to 0.04. Although our unbridged repeat resolution approach was not able to resolve these repeats, we still partially reconstruct the repeat copy sequence with high accuracy. *One simulation with repeat length 40 kb and divergence 0.04% did not generate repeat sequences for the unbridged repeat resolution tool to be applied so no results are shown.

length (kb)	cov	div (%)	rep	#true divergent positions	#confirmed divergent positions	maximal distance between positions (kb)	#linking reads	resolved	mean reconstructed sequence length (kb)	mean reconstructed sequence accuracy
20	50x	0.05	A	7	2	19.4	0	NO	13	99.92%
			B	9	11	11.9	0	NO	10	99.87%
		0.15	A	32	34	3.8	20	YES	20	99.97%
			B	35	42	2.3	21	YES	20	99.94%
		0.45	A	103	113	0.9	252	YES	20	99.95%
			B	81	90	1.5	289	YES	20	99.97%
40	50x	0.05	A	19	6	30.6	0	NO	8	99.87%
			B	18	4	34.8	0	NO	12	99.88%
		0.15	A	68	81	3.8	13	YES	40	99.95%
			B	64	75	2.9	35	YES	40	99.95%
		0.45	A	194	213	0.9	425	YES	40	99.96%
			B	169	189	1.5	411	YES	40	99.97%
20	30x	0.05	A	7	4	17.4	0	NO	10	99.87%
			B	9	6	16.9	0	NO	6	99.82%
		0.15	A	32	36	7.6	2	NO	19	99.89%
			B	35	39	2.9	7	YES	18	99.91%
		0.45	A	103	117	0.9	30	YES	20	99.93%
			B	81	94	1.5	23	YES	20	99.89%
40	30x	0.05	A	19	2	37.2	0	NO	5	99.85%
			B	18	4	36.7	0	NO	9	99.80%
		0.15	A	68	46	19.3	0	NO	23	99.90%
			B	64	9	36.9	0	NO	12	99.68%
		0.45	A	194	220	0.9	175	YES	40	99.91%
			B	169	192	1.0	208	YES	40	99.92%

Table S7. Unbridged repeat resolution low coverage simulation results. Genomes containing repeats of multiplicity two were simulated for low coverage rates of 50x and 30x. The other conditions were kept the same as Table S5. Even at lower coverage, most repeats above 0.1% divergence were able to be resolved.

SUPPLEMENTARY TABLES

dataset	bacterial species	Flye	Flye + unbridged repeat resolution	HINGE
EC4450	*Escherichia coli*	Tangled	n/a	Tangled
KP5052	*Klebsiella pneumoniae*	Tangled	Tangled	Tangled
SA6134	*Staphylococcus aureus*	Complete	n/a	Complete
EC7921	*Escherichia coli*	Tangled	Complete	Complete
EC8333	*Escherichia coli*	Tangled*	n/a	Tangled
EC8781	*Escherichia coli*	Tangled	n/a	Tangled
EC9002	*Escherichia coli*	Complete	n/a	Complete
EC9006	*Escherichia coli*	Tangled	Tangled	Tangled
EC9007	*Escherichia coli*	Tangled	Tangled	Tangled
EC9012	*Escherichia coli*	Tangled	Tangled	Complete
EC9016	*Escherichia coli*	Tangled	Tangled	Tangled
EC9024	*Escherichia coli*	Tangled	n/a	Tangled
EC9103	*Escherichia coli*	Complete	n/a	Complete
KP9657	*Klebsiella pneumoniae*	Tangled	n/a	Tangled
EC9664	*Escherichia coli*	Tangled	Complete	Tangled
EC10864	*Escherichia coli*	Tangled	n/a	Complete
EC11022	*Escherichia coli*	Tangled	Semi-complete	Complete
KS11692	*Klebsiella sp*	Tangled	n/a	Complete
SA11962	*Staphylococcus aureus*	Tangled	Tangled	Tangled
KP12158	*Klebsiella planticola*	Semi-complete	n/a	Complete
KC12993	*Kluyvera cryocrescens*	Complete	n/a	Complete

Supplementary Table 1. A comparison of Flye and HINGE on bacterial genomes from the BACTERIA dataset. HINGE results were reproduced from Kamath et al., 2017. “Tangled*” stands for “Tangled/Lacks Circularization”. “n/a” indicates that the assembly graph is not complete and has no unbridged repeats of multiplicity two.

dataset	repeat length (kb)	coverage	divergence	#tentative divergent positions	#confirmed divergent positions	maximal distance between divergent positions (kb)	remaining gap (kb)	*#cis* linking reads	*#trans* linking reads
EC4450-29	11	159x	7.33%	657	594	0.4	0	1219	42
KN5052-10	38	98x	1.12%	376	250	20.8	0	0	0
KN5052-20	31	96x	2.67%	826	676	17.3	0	1	0
EC7921-6	13	82x	11.51%	1629	1338	0.3	0	215	814
EC9002-3	50	137x	5.91%	3401	3064	0.9	0	2460	437
EC9006-8	22	94x	1.24%	218	131	11.7	0	0	0
EC9006-9	14	78x	2.81%	676	597	1.8	0	256	15
EC9006-10	16	93x	5.25%	2843	2610	0.8	0	912	80
EC9007-5	24	140x	0.33%	2467	37	14.6	5.0	0	0
EC9012-7	14	63x	19.22%	2784	2552	1.3	0	599	42
EC9012-12	37	74x	3.12%	1973	1601	2.0	0	1126	13
EC9016-4	17	47x	8.45%	2586	2340	2.4	0	462	34
EC9016-5	24	58x	1.30%	1210	203	21.5	0.3	0	0
EC9103-4	4	131x	6.62%	340	314	0.4	0	135	87
KN9657-9	36	61x	0.08%	186	3	35.7	4.3	0	0
EC9964-5	34	73x	6.20%	2333	2179	0.9	0	64	892
EC9964-6	22	80x	4.17%	1675	1522	1.7	0	12	649
EC11022-7	30	60x	1.44%	1661	1491	6.3	0	2	602
EC11022-8	25	64x	0.37%	165	17	10.1	0	0	0
SA11962-6	8	159x	11.39%	613	562	0.5	0	1089	16
SA11962-8*	13	214x	0.77%	154	100	4.1	0	40	42
KL12158-7	13	46x	0.06%	50	0	12.7	0	0	0

Supplementary Table 2. Resolving unbridged repeats of multiplicity two in genomes from the BACTERIA dataset. The results of repeat resolution after running Flye for 11 out of 21 genomes from the BACTERIA datasets that contain repeats of multiplicity two. The label of each dataset denotes the bacterial species, its strain, and the ID number of the repeat edge found in the assembly graph (e.g. EC5052-7, EC5052-8, and EC5052-9 refer to 3 repeats with IDs "7", "8", and "9" present in the assembly graph for the *E. coli* NCTC5052 dataset). Bolded labels refer to repeats resolved by Flye. The * refers to a repeat of multiplicity 3. The "coverage" is calculated as the total read length divided by the repeat length, divided by the multiplicity of the repeat (comparable to the coverage of a normal genomic sequence of multiplicity one). The "divergence" is calculated based on the alignment of constructed repeat consensus sequences, dividing the total number of substitutions and indels by the total number of matches, substitutions, and indels (if the forward and reverse consensus sequences do not overlap, then the mean divergence of the forward and reverse sequences is calculated, weighted by the length of the sequences). "Maximal distance between divergent positions" refers to the maximal distance between adjacent confirmed divergent positions. "Remaining gap" refers to the length of the repeat remaining without separate consensus sequences for each copy after we have "moved into the repeat" from both the forward and reverse directions (note that it is 0 if the forward and reverse consensus sequences overlap).

REFERENCES

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).

Bankevich, A. *et al*. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.* **19**, 455-477 (2012).

Bankevich A. & Pevzner, P.A. TruSPAdes: barcode assembly of TruSeq synthetic long reads. *Nat. Methods* **13**, 248–250 (2015)

Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 34, 3094-3100 (2018)

Lin, Y. *et al*. Assembly of long error-prone reads using de Bruijn graphs. *Proc. Nat. Acad. Sci. USA*. **113**, E8396-E8405 (2016).

Myers, E.W. *et al.* A whole-genome assembly of Drosophila. *Science* **287**, 2196–2204 (2000)

Nurk, S., Meleshko, D., Korobeynikov, A. & Pevzner, P.A. metaSPAdes: a new versatile metagenomic assembler. *Genome Res.* **27**, pp.824-834 (2017)

Ono, Y, Asai, K. & Hamada, M. PBSIM: PacBio reads simulator—toward accurate genome assembly *Bioinformatics* **29**, 1 119–121 (2013)

Pevzner, P.A., Tang, H. & Tesler, G. De novo repeat classification and fragment assembly. *Genome Res.* **14**, 1786-1796 (2004).

Prjibelski, A. D. *et al*. ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics* **30**, i293-i301 (2014)

Sharon I. *et al*. Accurate, multi-kb reads resolve complex populations and detect rare microorganisms. *Genome Res.* **25**, 534–543 (2015)

Treangen, T.J. *et al.* MetAMOS: a modular and open source metagenomic assembly and analysis pipeline. *Genome Biology* **14**, R2 (2013).

Vyahhi, N., Pyshkin, A., Pham, S., & Pevzner, P.A. From de Bruijn graphs to rectangle graphs for genome assembly. *Lect. Notes Comput. Sc.* **7534**, 249-261 (2012)

Wick R. R., Judd L. M., Gorrie C. L. & Holt K.E. Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Comput. Biol.* **13**, e1005595 (2017)