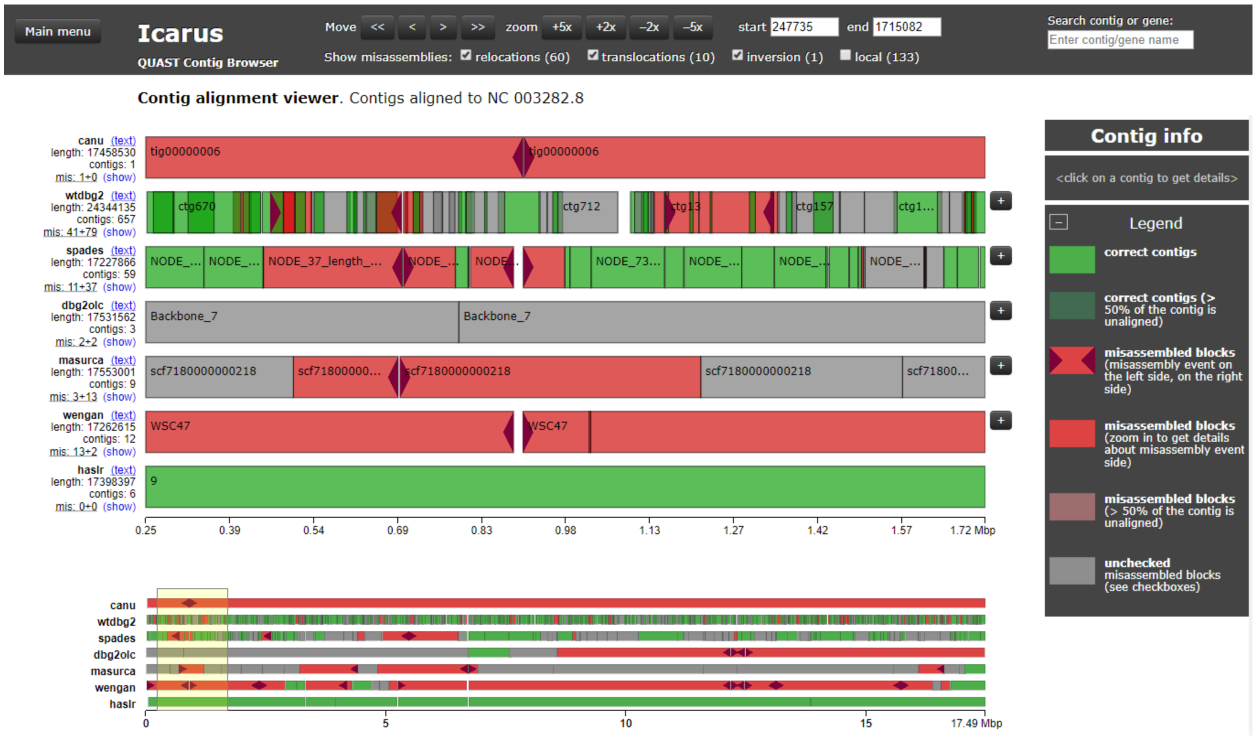
iScience, Volume *23*

Supplemental Information

HASLR: Fast Hybrid Assembly of Long Reads

Ehsan Haghshenas, Hossein Asghari, Jens Stoye, Cedric Chauve, and Faraz Hach

S1 Supplemental Figures



start 247735 end 1715082

Show misassemblies: ☑ relocations (60) ☑ translocations (10) ☑ inversion (1) ☐ local (133)

Contig alignment viewer. Contigs aligned to NC 003282.8

canu (text) length: 17458530 contigs: 1 mis: 1+0 (show)

wtdbg2 (text) length: 24344135 contigs: 657 mis: 41+79 (show)

spades (text) length: 17227866 contigs: 59 mis: 11+37 (show)

dbg2olc (text) length: 17531562 contigs: 3 mis: 2+2 (show)

masurca (text) length: 17553001 contigs: 9 mis: 3+13 (show)

wengan (text) length: 17262615 contigs: 12 mis: 13+2 (show)

haslr (text) length: 17398397 contigs: 6 mis: 0+0 (show)

🟩 **correct contigs**

🟩 **correct contigs** (> 50% of the contig is unaligned)

🟥 **misassembled blocks** (misassembly event on the left side, on the right side)

🟥 **misassembled blocks** (zoom in to get details about misassembly event side)

🟥 **misassembled blocks** (> 50% of the contig is unaligned)

⬜ **unchecked** misassembled blocks (see checkboxes)

Figure S1. An example showing a region of choromosome 4 of *C. elegans*. Related to Table 1.

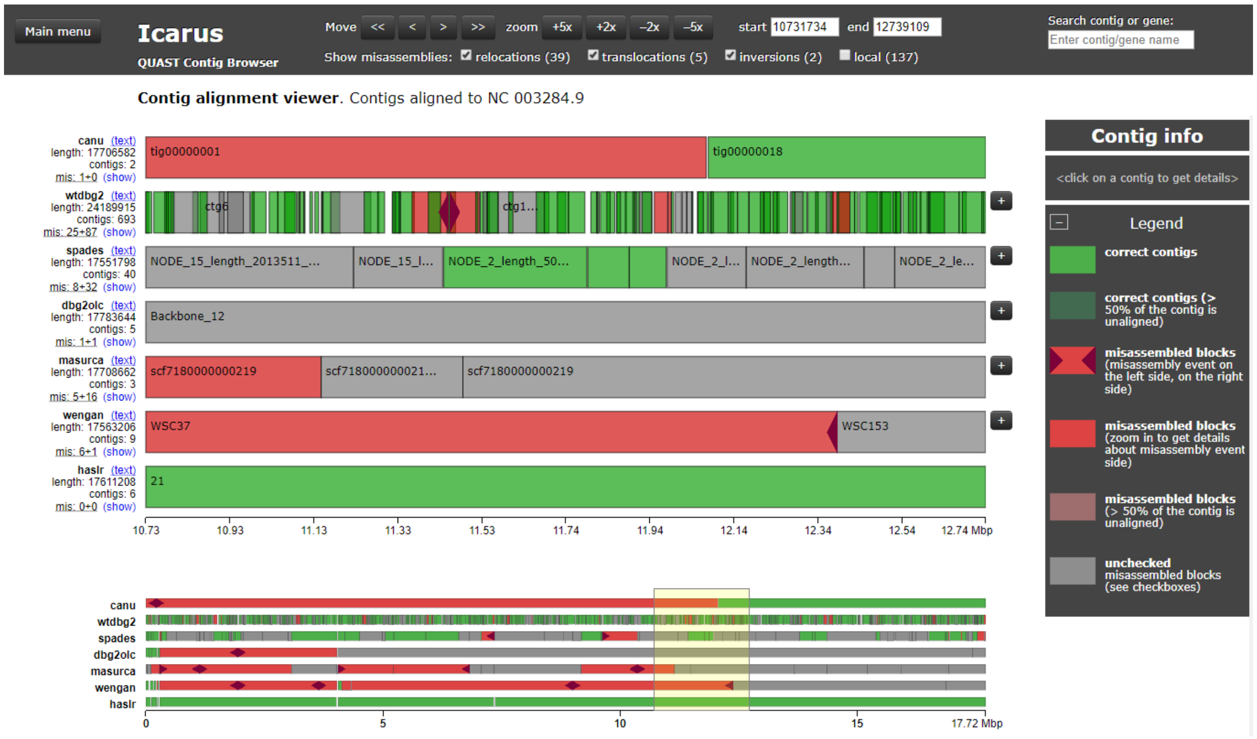

Main menu
Icarus
QUAST Contig Browser
Move << < > >> zoom +5x +2x −2x −5x start 10731734 end 12739109
Show misassemblies: ☑ relocations (39) ☑ translocations (5) ☑ inversions (2) ☐ local (137)
Search contig or gene:
Enter contig/gene name
Contig alignment viewer. Contigs aligned to NC 003284.9
canu (text) length: 17706582 contigs: 2 mis: 1+0 (show)
wtdbg2 (text) length: 24189915 contigs: 693 mis: 25+87 (show)
spades (text) length: 17551798 contigs: 40 mis: 8+32 (show)
dbg2olc (text) length: 17783644 contigs: 5 mis: 1+1 (show)
masurca (text) length: 17708662 contigs: 3 mis: 5+16 (show)
wengan (text) length: 17563206 contigs: 9 mis: 6+1 (show)
haslr (text) length: 17611208 contigs: 6 mis: 0+0 (show)
tig00000001
tig00000018
ctg6
ctg1...
NODE_15_length_2013511_...
NODE_15_l...
NODE_2_length_50...
NODE_2_l...
NODE_2_length...
NODE_2_le...
Backbone_12
scf7180000000219
scf718000000021...
scf7180000000219
WSC37
WSC153
21
10.73 10.93 11.13 11.33 11.53 11.74 11.94 12.14 12.34 12.54 12.74 Mbp
canu
wtdbg2
spades
dbg2olc
masurca
wengan
haslr
0 5 10 15 17.72 Mbp
Contig info
<click on a contig to get details>
Legend
correct contigs
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S2. An example showing a region of choromosome X of *C. elegans*. Related to Table 1.

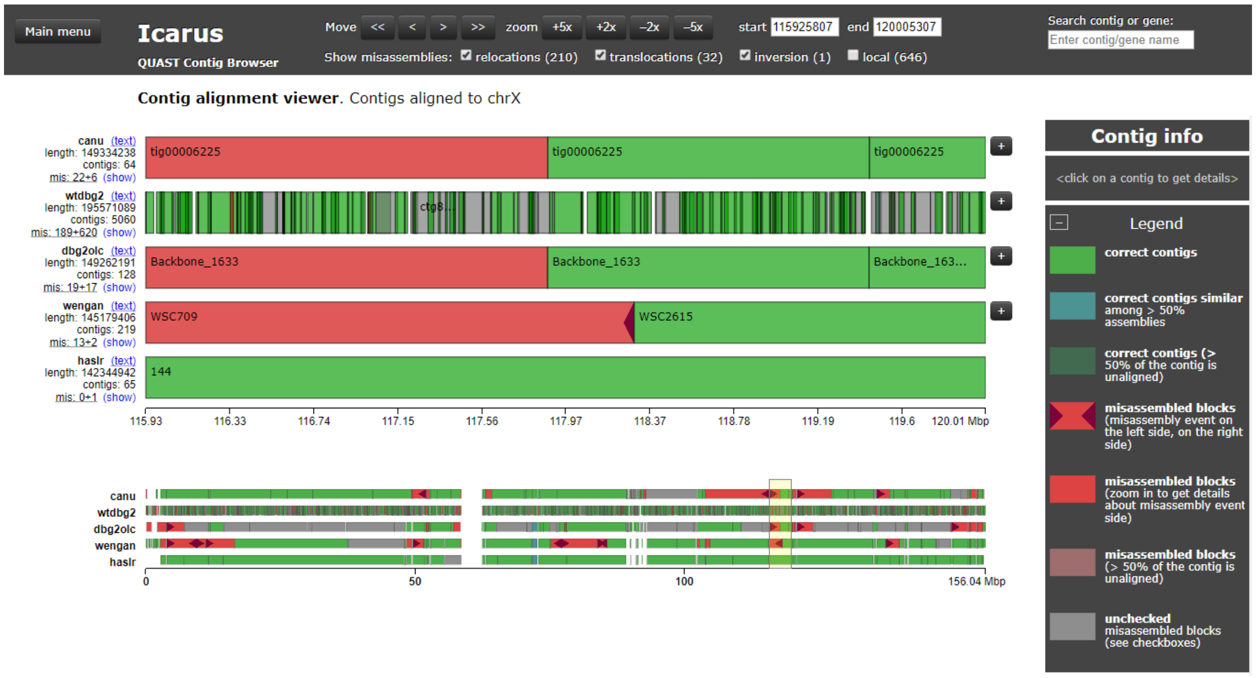
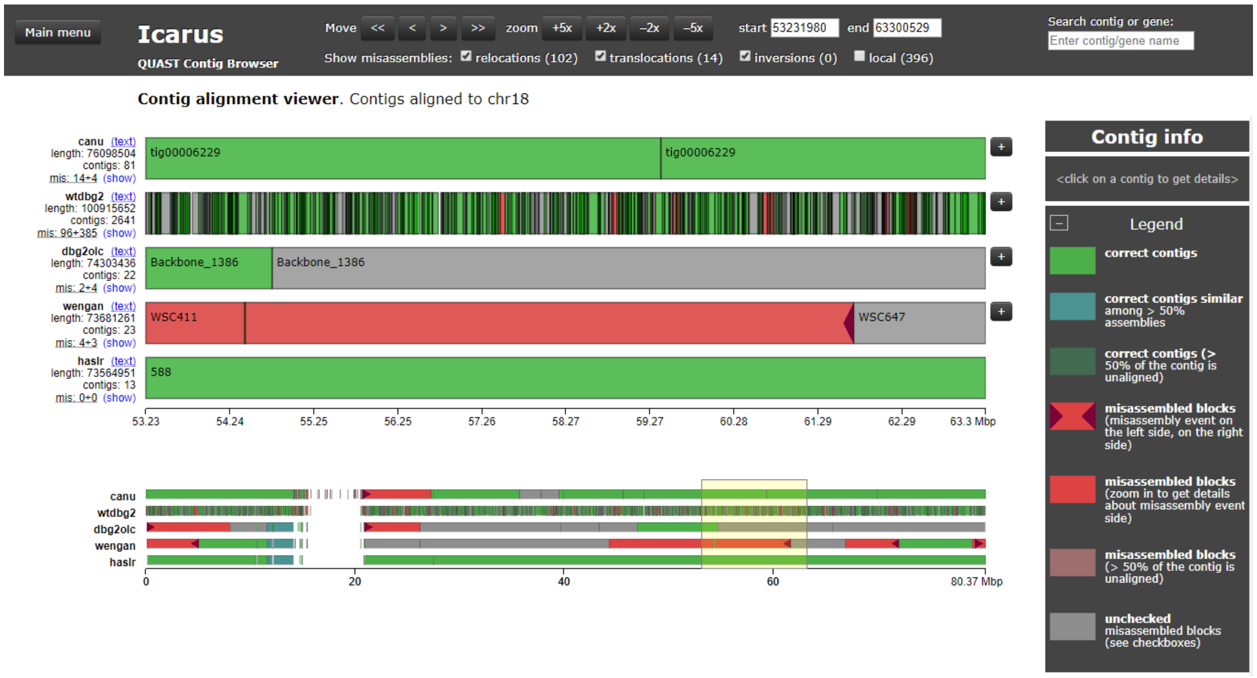

Main menu
Icarus
QUAST Contig Browser
Move << < > >> zoom +5x +2x −2x −5x start 115925807 end 120005307
Show misassemblies: ☑ relocations (210) ☑ translocations (32) ☑ inversion (1) ☐ local (646)
Search contig or gene:
Enter contig/gene name
Contig alignment viewer. Contigs aligned to chrX
canu (text) length: 149334238 contigs: 64 mis: 22+6 (show)
wtdbg2 (text) length: 195571089 contigs: 5060 mis: 189+620 (show)
dbg2olc (text) length: 149262191 contigs: 128 mis: 19+17 (show)
wengan (text) length: 145179406 contigs: 219 mis: 13+2 (show)
haslr (text) length: 142344942 contigs: 65 mis: 0+1 (show)
tig00006225
tig00006225
tig00006225
ctg8...
Backbone_1633
Backbone_1633
Backbone_163...
WSC709
WSC2615
144
115.93 116.33 116.74 117.15 117.56 117.97 118.37 118.78 119.19 119.6 120.01 Mbp
canu
wtdbg2
dbg2olc
wengan
haslr
0 50 100 156.04 Mbp
Contig info
<click on a contig to get details>
Legend
correct contigs
correct contigs similar among > 50% assemblies
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S3. An example showing a region of choromosome X of hg38. Related to Table 1.



Main menu

Icarus

QUAST Contig Browser

Move << < > >> zoom +5x +2x −2x −5x start 53231980 end 63300529

Show misassemblies: ☑ relocations (102) ☑ translocations (14) ☑ inversions (0) ☐ local (396)

Search contig or gene: Enter contig/gene name

Contig alignment viewer. Contigs aligned to chr18

canu (text) length: 76098504 contigs: 81 mis: 14+4 (show)

wtdbg2 (text) length: 100915652 contigs: 2641 mis: 96+385 (show)

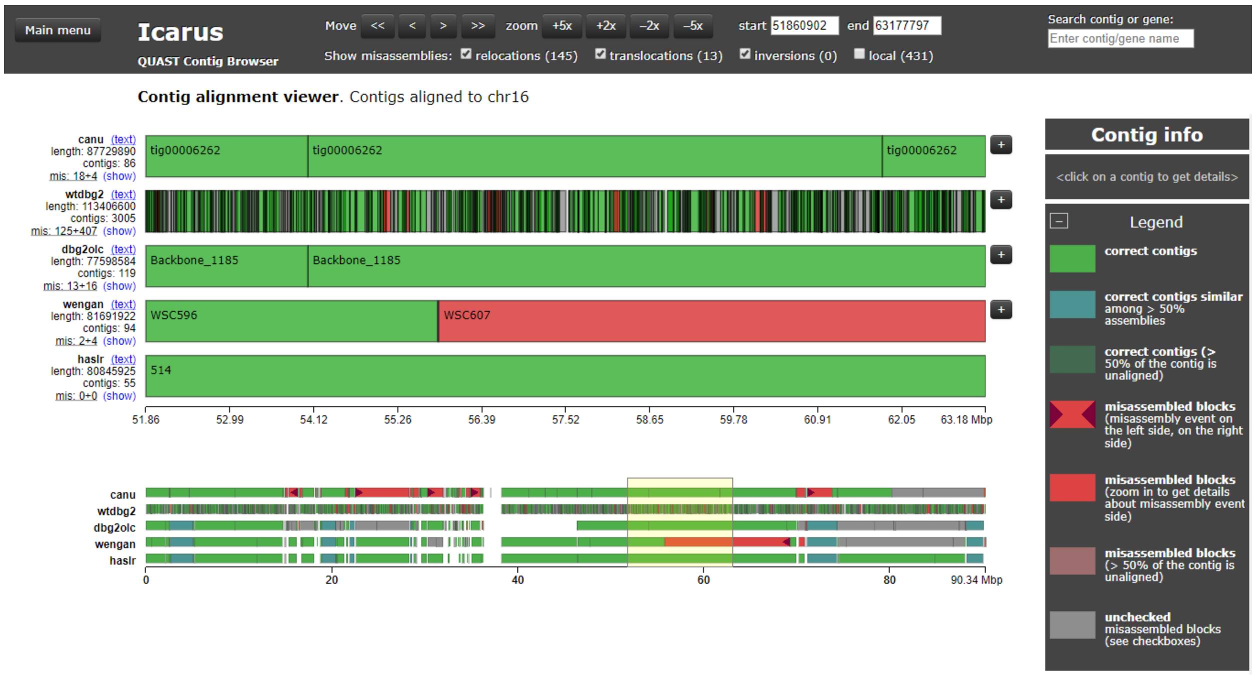
dbg2olc (text) length: 74303436 contigs: 22 mis: 2+4 (show)

wengan (text) length: 73681261 contigs: 23 mis: 4+3 (show)

hasIr (text) length: 73564951 contigs: 13 mis: 0+0 (show)

tig00006229
tig00006229
Backbone_1386
Backbone_1386
WSC411
WSC647
588
53.23
54.24
55.25
56.25
57.26
58.27
59.27
60.28
61.29
62.29
63.3 Mbp
canu
wtdbg2
dbg2olc
wengan
hasIr
0
20
40
60
80.37 Mbp
Contig info
<click on a contig to get details>
Legend
🟩 correct contigs
correct contigs similar among > 50% assemblies
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
🟥 misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S4. An example showing a region of choromosome 18 of hg38. Related to Table 1.



Main menu

Icarus

QUAST Contig Browser

Move << < > >> zoom +5x +2x −2x −5x start 51860902 end 63177797

Show misassemblies: ☑ relocations (145) ☑ translocations (13) ☑ inversions (0) ☐ local (431)

Search contig or gene: Enter contig/gene name

Contig alignment viewer. Contigs aligned to chr16

canu (text) length: 87729890 contigs: 86 mis: 18+4 (show)

wtdbg2 (text) length: 113406600 contigs: 3005 mis: 125+407 (show)

dbg2olc (text) length: 77598584 contigs: 119 mis: 13+16 (show)

wengan (text) length: 81691922 contigs: 94 mis: 2+4 (show)

hasIr (text) length: 80845925 contigs: 55 mis: 0+0 (show)

tig00006262
tig00006262
tig00006262
Backbone_1185
Backbone_1185
WSC596
WSC607
514
51.86
52.99
54.12
55.26
56.39
57.52
58.65
59.78
60.91
62.05
63.18 Mbp
canu
wtdbg2
dbg2olc
wengan
hasIr
0
20
40
60
80
90.34 Mbp
Contig info
<click on a contig to get details>
Legend
🟩 correct contigs
correct contigs similar among > 50% assemblies
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
🟥 misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S5. An example showing a region of choromosome 16 of hg38. Related to Table 1.

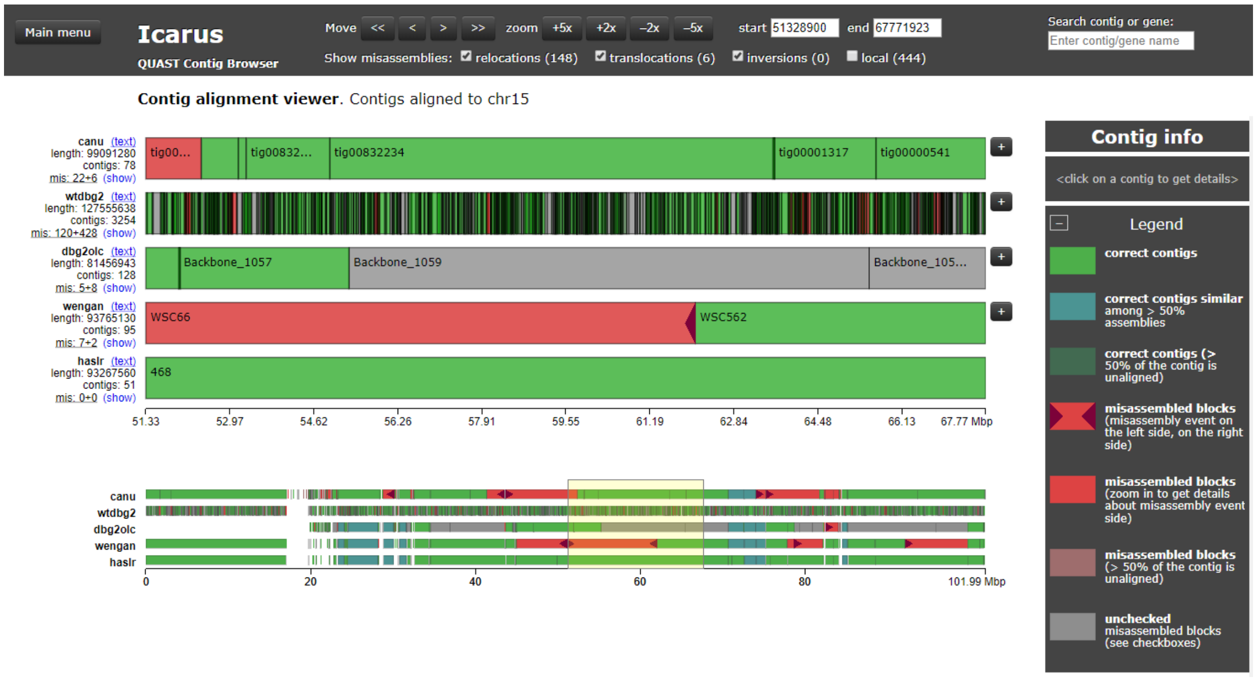

Main menu
Icarus
QUAST Contig Browser
Move << < > >> zoom +5x +2x −2x −5x start 51328900 end 67771923
Show misassemblies: relocations (148) translocations (6) inversions (0) local (444)
Search contig or gene:
Enter contig/gene name
Contig alignment viewer. Contigs aligned to chr15
canu (text) length: 99091280 contigs: 78 mis: 22+6 (show)
wtdbg2 (text) length: 127555638 contigs: 3254 mis: 120+428 (show)
dbg2olc (text) length: 81456943 contigs: 128 mis: 5+8 (show)
wengan (text) length: 93765130 contigs: 95 mis: 7+2 (show)
haslr (text) length: 93267560 contigs: 51 mis: 0+0 (show)
tig00...
tig00832...
tig00832234
tig00001317
tig00000541
Backbone_1057
Backbone_1059
Backbone_105...
WSC66
WSC562
468
51.33
52.97
54.62
56.26
57.91
59.55
61.19
62.84
64.48
66.13
67.77 Mbp
canu
wtdbg2
dbg2olc
wengan
haslr
0
20
40
60
80
101.99 Mbp
Contig info
<click on a contig to get details>
Legend
correct contigs
correct contigs similar among > 50% assemblies
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S6. An example showing a region of choromosome 15 of hg38. Related to Table 1.

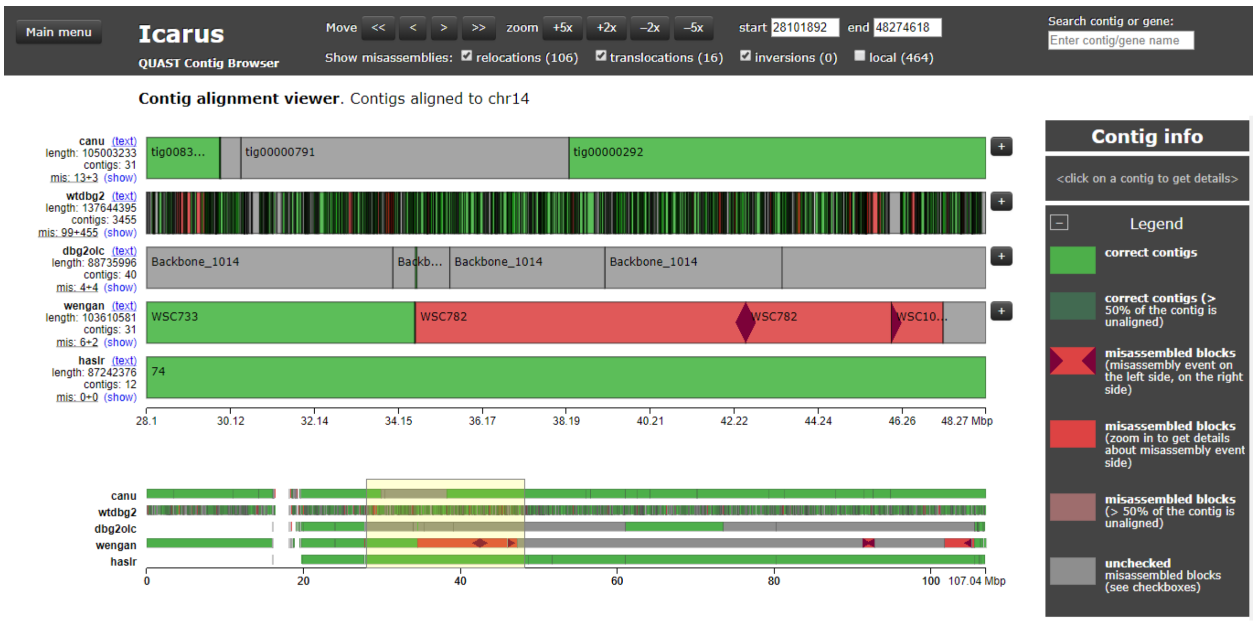
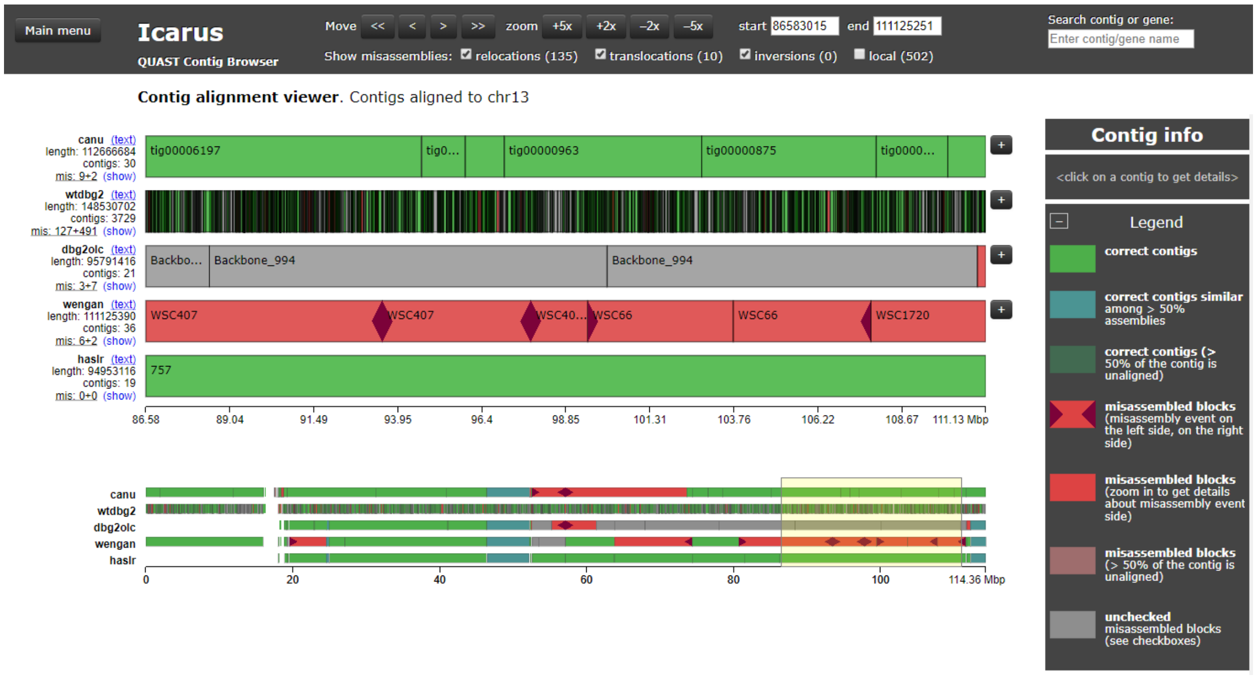

Main menu
Icarus
QUAST Contig Browser
Move << < > >> zoom +5x +2x −2x −5x start 28101892 end 48274618
Show misassemblies: relocations (106) translocations (16) inversions (0) local (464)
Search contig or gene:
Enter contig/gene name
Contig alignment viewer. Contigs aligned to chr14
canu (text) length: 105003233 contigs: 31 mis: 13+3 (show)
wtdbg2 (text) length: 137644395 contigs: 3455 mis: 99+455 (show)
dbg2olc (text) length: 88735996 contigs: 40 mis: 4+4 (show)
wengan (text) length: 103610581 contigs: 31 mis: 6+2 (show)
haslr (text) length: 87242376 contigs: 12 mis: 0+0 (show)
tig0083...
tig00000791
tig00000292
Backbone_1014
Backb...
Backbone_1014
Backbone_1014
WSC733
WSC782
WSC782
WSC10...
74
28.1
30.12
32.14
34.15
36.17
38.19
40.21
42.22
44.24
46.26
48.27 Mbp
canu
wtdbg2
dbg2olc
wengan
haslr
0
20
40
60
80
100
107.04 Mbp
Contig info
<click on a contig to get details>
Legend
correct contigs
correct contigs (> 50% of the contig is unaligned)
misassembled blocks (misassembly event on the left side, on the right side)
misassembled blocks (zoom in to get details about misassembly event side)
misassembled blocks (> 50% of the contig is unaligned)
unchecked misassembled blocks (see checkboxes)

Figure S7. An example showing a region of choromosome 14 of hg38. Related to Table 1.


Main menu

Icarus
QUAST Contig Browser

Move << < > >> zoom +5x +2x −2x −5x start 86583015 end 111125251

Show misassemblies: ☑ relocations (135) ☑ translocations (10) ☑ inversions (0) ☐ local (502)

Search contig or gene: Enter contig/gene name

Contig alignment viewer. Contigs aligned to chr13

canu (text)
length: 112666684
contigs: 30
mis: 9+2 (show)

wtdbg2 (text)
length: 148530702
contigs: 3729
mis: 127+491 (show)

dbg2olc (text)
length: 95791416
contigs: 21
mis: 3+7 (show)

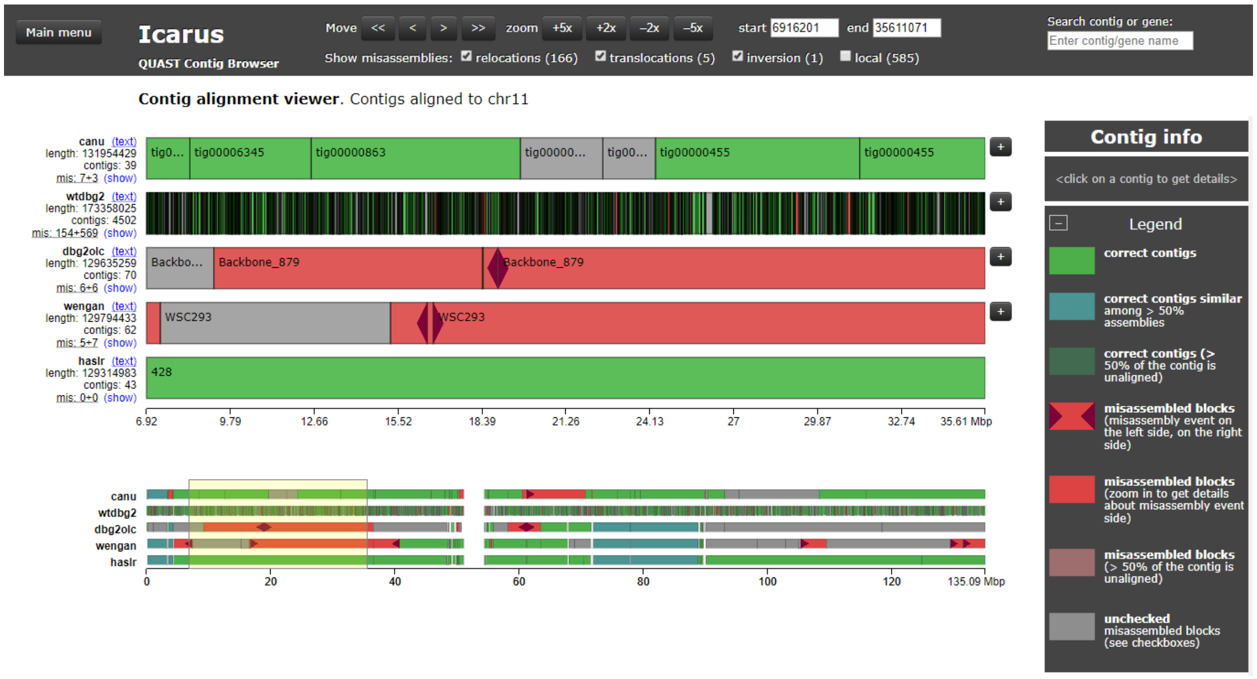
wengan (text)
length: 111125390
contigs: 36
mis: 6+2 (show)

haslr (text)
length: 94953116
contigs: 19
mis: 0+0 (show)

Contig info
<click on a contig to get details>

Legend
🟩 correct contigs
🟦 correct contigs similar among > 50% assemblies
🟩 correct contigs (> 50% of the contig is unaligned)
🟥 misassembled blocks (misassembly event on the left side, on the right side)
🟥 misassembled blocks (zoom in to get details about misassembly event side)
🟥 misassembled blocks (> 50% of the contig is unaligned)
⬜ unchecked misassembled blocks (see checkboxes)

Figure S8. An example showing a region of choromosome 13 of hg38. Related to Table 1.


Main menu

Icarus
QUAST Contig Browser

Move << < > >> zoom +5x +2x −2x −5x start 6916201 end 35611071

Show misassemblies: ☑ relocations (166) ☑ translocations (5) ☑ inversion (1) ☐ local (585)

Search contig or gene: Enter contig/gene name

Contig alignment viewer. Contigs aligned to chr11

canu (text)
length: 131954429
contigs: 39
mis: 7+3 (show)

wtdbg2 (text)
length: 173358025
contigs: 4502
mis: 154+569 (show)

dbg2olc (text)
length: 129635259
contigs: 70
mis: 6+6 (show)

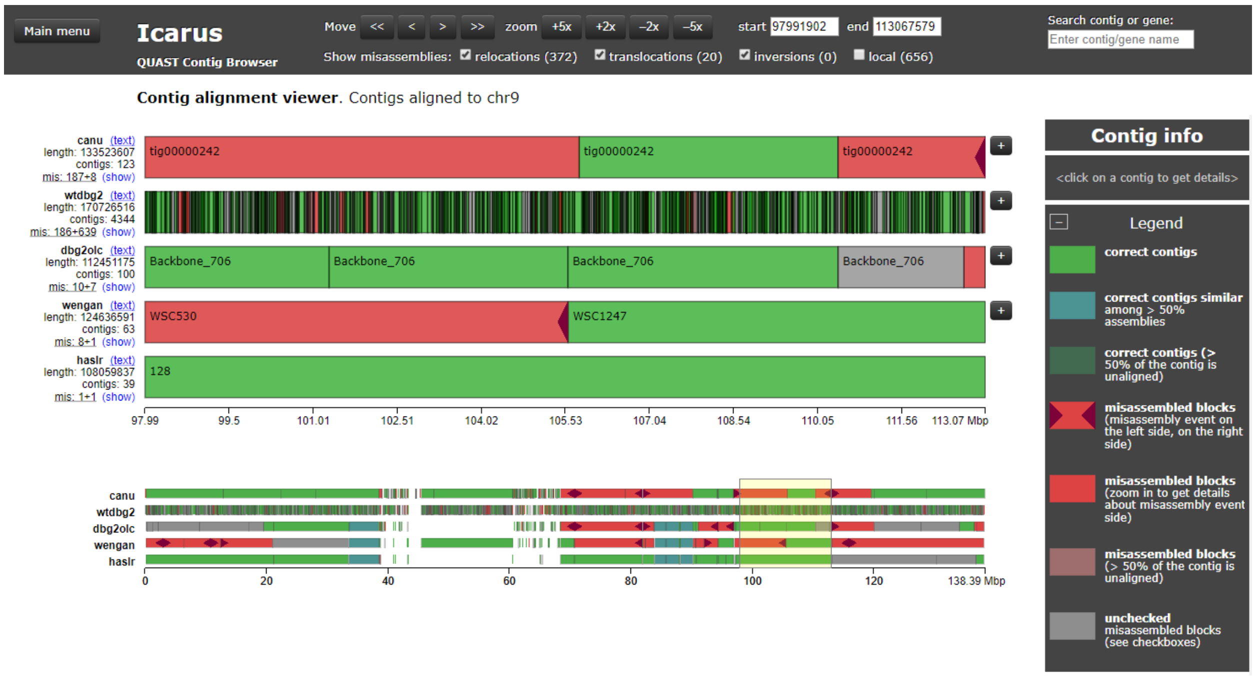
wengan (text)
length: 129794433
contigs: 62
mis: 5+7 (show)

haslr (text)
length: 129314983
contigs: 43
mis: 0+0 (show)

Contig info
<click on a contig to get details>

Legend
🟩 correct contigs
🟦 correct contigs similar among > 50% assemblies
🟩 correct contigs (> 50% of the contig is unaligned)
🟥 misassembled blocks (misassembly event on the left side, on the right side)
🟥 misassembled blocks (zoom in to get details about misassembly event side)
🟥 misassembled blocks (> 50% of the contig is unaligned)
⬜ unchecked misassembled blocks (see checkboxes)

Figure S9. An example showing a region of choromosome 11 of hg38. Related to Table 1.


Icarus

QUAST Contig Browser

start 97991902 end 113067579

Show misassemblies: ☑ relocations (372) ☑ translocations (20) ☑ inversions (0) ☐ local (656)

Contig alignment viewer. Contigs aligned to chr9

canu (text)
length: 133523607
contigs: 123
mis: 187+8 (show)

wtdbg2 (text)
length: 170726516
contigs: 4344
mis: 186+639 (show)

dbg2olc (text)
length: 112451175
contigs: 100
mis: 10+7 (show)

wengan (text)
length: 124636591
contigs: 63
mis: 8+1 (show)

haslr (text)
length: 108059837
contigs: 39
mis: 1+1 (show)

Contig info

<click on a contig to get details>

Legend

🟩 **correct contigs**

🟦 **correct contigs similar** among > 50% assemblies

🟩 **correct contigs (>** 50% of the contig is unaligned)

🟥 **misassembled blocks** (misassembly event on the left side, on the right side)

🟥 **misassembled blocks** (zoom in to get details about misassembly event side)

🟫 **misassembled blocks** (> 50% of the contig is unaligned)

⬜ **unchecked** misassembled blocks (see checkboxes)

Figure S10. An example showing a region of choromosome 9 of hg38. Related to Table 1.

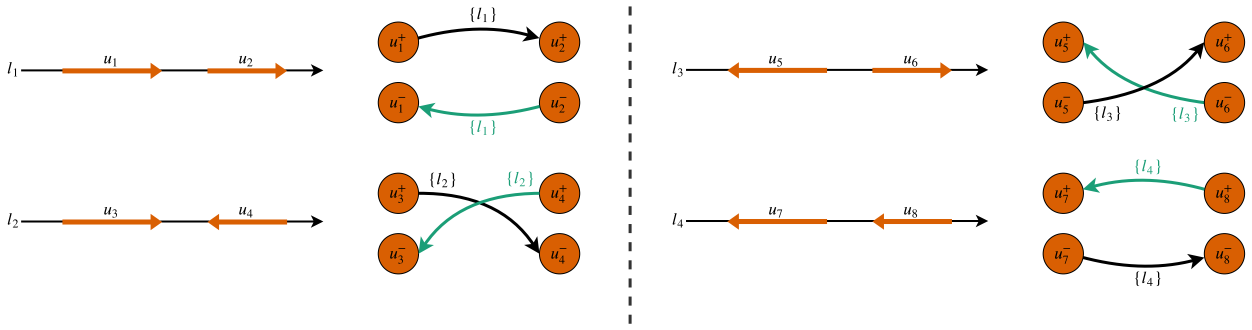
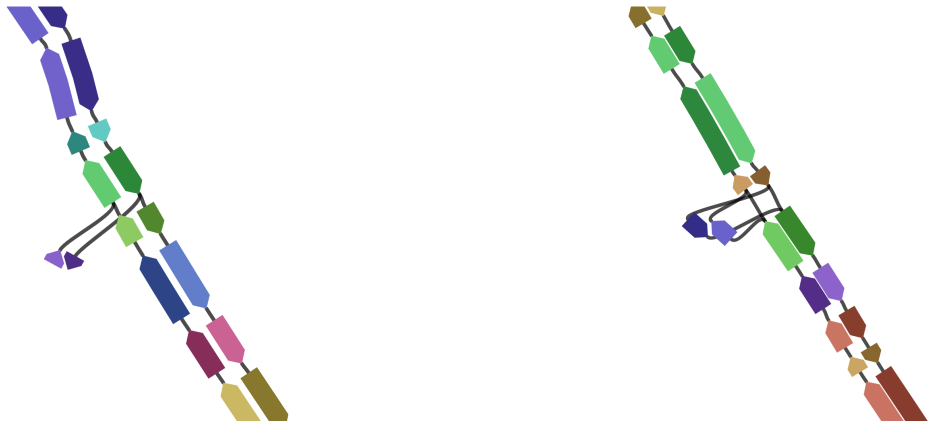


Figure S11. Possible orientations of aligning two unique contigs to a long read. The direction of contigs aligned to long reads shows the strand of their corresponding sequence. These directions guide us to find the proper edge type. The set of long reads supporting each edge is shown as its label. Related to Figure 1.



(a) example of a tip in the backbone graph

(b) example of a bubble in the backbone graph

Figure S12. Examples of tip and bubbles in the backbone graph. Here the backbone graph is visualized using Bandage (Wick et al., 2015). Related to Figure 1.

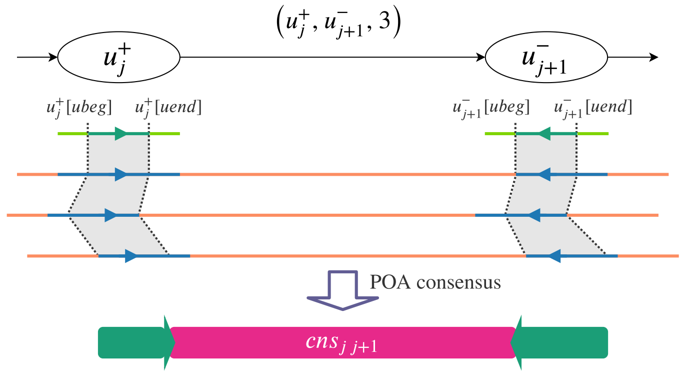


Figure S13. Example of an edge in backbone graph and its corresponding long read alignments. Partial Order Alignment (POA) is used in constructing the consensus sequence (see subsection S3.5). Related to Figure 1.

S2 Supplemental Tables

Table S1: Details about utilized software. Related to Tables 1 and 3.

Tool	Version	Reference	Repository
Minia	3.2.1	Chikhi and Rizk (2013)	`github.com/GATB/minia`
minimap2	2.17	Li (2018)	`github.com/lh3/minimap2`
SPOA	1.1.3	Vaser et al. (2017)	`github.com/rvaser/spoa`
GNU Time	1.9	–	`ftp.gnu.org/gnu/time/`
ART	2.5.8	Huang et al. (2011)	`niehs.nih.gov/research/resources/software/biostatistics/art/`
PBSIM	7fdcefd	Ono et al. (2012)	`github.com/yukiteruono/pbsim`
Canu	1.8	Koren et al. (2017)	`github.com/marbl/canu`
Flye	2.6	Kolmogorov et al. (2019)	`github.com/fenderglass/Flye`
wtdbg2	2.5	Ruan and Li (2019)	`github.com/ruanjue/wtdbg2`
miniasm	0.3	Li (2016)	`https://github.com/lh3/miniasm`
SPAdes	3.13.1	Antipov et al. (2015)	`github.com/ablab/spades`
Unicycler	0.4.8	Wick et al. (2017)	`github.com/rrwick/unicycler`
DBG2OLC	0246e46	Ye et al. (2016)	`github.com/yechengxi/dbg2olc`
MaSuRCA	3.3.1	Zimin et al. (2017)	`github.com/alekseyzimin/masurca`
Wengan	v0.1	Di Genova et al. (2019)	`github.com/adigenova/wengan`
QUAST	5.0.2	Mikheenko et al. (2018)	`github.com/ablab/quast`
BUSCO	4.0.1	Simão et al. (2015)	`busco.ezlab.org`

Table S2: Comparison between assemblies obtained by different tools on HG002 dataset against human reference GRCh38. Related to Table 3.

Assembler	*Contigs*	*Genome fraction*	*NGA50*	*Misassemblies extensive+local*	*Mismatch rate*	*Indel rate*	*Time*	*Memory (GB)*
Canu	6,227	96.203	1,832,773	6,145+7,285	136.16	79.05	533:25:31	34.31
Flye	NA							
wtdbg2	4,768	93.935	2,084,440	3,200+6,320	111.72	97.05	12:24:45	211.56
miniasm	5,762	95.537	1,463,623	3,222+10,145	162.20	575.98	94:12:20	444.65
Minia	575,982	84.428	4,694	1,374+1,518	83.65	16.99	9:22:10	8.66
SPAdes	NA							
hybridSPAdes	NA							
Unicycler	NA							
DBG2OLC	NA							
MaSuRCA	NA							
Wengan	2867	93.297	1,217,282	2,455+7,034	108.93	82.97	34:51:29	49.36
HASLR	11,557	92.487	424,477	2,397+8,908	113.94	209.92		

Note: Mismatch and indel rates are reported per 100 kbp. Flye, SPAdes, hybridSPAdes, and Unicycler failed due to memory limit. DBG2OLC failed due to exceeding the limit for the number of open files in the cluster (4000). MaSuRCA crashed after running for 40 days.

Table S3: Comparison between assemblies obtained by different tools on HG002 dataset against Peregrine assembly of HiFi PacBio reads. Related to Table 3.

Assembler	Contigs	Genome fraction	NGA50	Misassemblies extensive+local	Mismatch rate	Indel rate	Time	Memory (GB)
Canu	6,227	93.394	3,344,052	3,828+3,324	75.63	74.23	533:25:31	34.31
Flye	NA							
wtdbg2	4,768	91.377	4,050,425	2,578+2,510	54.90	93.36	12:24:45	211.56
miniasm	5,762	92.676	2,421,361	2,237+6,387	109.24	577.53	94:12:20	444.65
Minia	575,982	81.771	4,826	1,210+616	31.05	8.53	9:22:10	8.66
SPAdes	NA							
hybridSPAdes	NA							
Unicycler	NA							
DBG2OLC	NA							
MaSuRCA	NA							
Wengan	2867	90.458	1,727,800	1,227+3,428	55.16	76.26	34:51:29	49.36
HASLR	11,557	89.624	495,840	1,129+5,019	58.87	204.59		

Note: Mismatch and indel rates are reported per 100 kbp. Flye, SPAdes, hybridSPAdes, and Unicycler failed due to memory limit. DBG2OLC failed due to exceeding the limit for the number of open files in the cluster (4000). MaSuRCA crashed after running for 40 days.

Table S4: Effect of polishing assemblies on the small assembly errors of two real datasets. Related to Table 3.

		Mismtach rate		Indel rate	
Dataset	Assembler	draft	polished	draft	polished
Yeast (PacBio)	Canu	8.85	7.56	7.99	7.99
	Flye	11.60	7.51	28.41	4.38
	wtdbg2	10.65	7.19	27.17	2.61
	miniasm	31.45	12.57	381.55	38.79
	hybridSPAdes	44.77	9.88	3.71	3.93
	Unicycler	15.13	6.84	4.22	2.44
	DBG2OLC	28.37	14.42	58.43	5.51
	MaSuRCA	11.83	8.49	5.85	9.69
	Wengan	11.86	7.36	34.29	2.08
	HASLR	8.13	4.33	100.64	2.05
C.elegans (PacBio)	Canu	65.28	65.88	58.82	29.71
	Flye	50.50	44.72	52.89	26.25
	wtdbg2	26.82	25.9	79.72	27.11
	miniasm	79.10	52.41	393.94	38.52
	hybridSPAdes	108.04	27.88	15.96	45.43
	Unicycler	58.36	36.97	45.47	32.08
	DBG2OLC	44.75	46.50	80.61	43.52
	MaSuRCA	49.20	30.9	23.50	31.97
	Wengan	35.75	21.13	121.11	22.82
	HASLR	26.08	19.61	140.40	22.92

Note: Here polished genomes are obtained after a single round of polishing using Arrow (`github.com/PacificBiosciences/GenomicConsensus`)

S3 Transparent Methods

S3.1 Obtaining unique short read contigs

The input to HASLR is a set of long reads (LRs) and a set of short reads (SRs) from the same sample, together with an estimation of the genome size. HASLR starts by assembling SRs into a set of *short read contigs*, denoted by C. Assembly of SRs is a well-studied topic and many efficient tools have been specifically designed for that purpose. These tools use either a de Bruijn graph (Simpson et al., 2009; Chikhi and Rizk, 2013) or an OLC strategy (based on an overlap graph or a string graph) (Simpson and Durbin, 2012; Molnar et al., 2017) to assemble the genome by finding "proper" paths in these graphs.

Next, HASLR identifies a set U of unique contigs (UCs), those SR contigs that are likely to appear in the genome only once. The motivation for this is that repetitive SR contigs would cause branching in the backbone graph and in fact, building the backbone graph using all SR contigs could result in a very tangled graph. In other words, using only unique SR contigs for building the backbone graph resolves many of the complexities and ambiguities in the graph. In order to identify unique contigs, for every SR contig, c_i, the mean k-mer frequency, $f(c_i)$, is computed as the average k-mer count of all k-mers present in c_i. Note that the value of $f(c_i)$ is proportional to the depth of coverage of c_i. Assuming longer contigs are more likely to come from unique regions, their mean k-mer frequency can be a good indicator for identifying UCs. Let $LC_q \subseteq C$ be the set of q longest SR contigs in C, and f_{avg}, f_{std} be the average and standard deviation of $\{f(c) \mid c \in LC_q\}$. Then, the set of unique contigs is defined as $U = \{u \mid u \in C \text{ and } f(u) \leq f_{avg} + 3f_{std}\}$. Our empirical results show that this approach can identify UCs with high precision and recall (see Section 2.2 for more details).

S3.2 Construction of backbone graph

The backbone graph encodes potential adjacencies between unique contigs and thus presents a large-scale map of the genome, albeit, with some level of ambiguity. Using the backbone graph, HASLR finds paths of unique contigs representing their relative order and orientation in the sequenced genome. These paths are later transformed into the assembly.

Formally, given a set of UCs, $U = \{u_1, u_2, \ldots, u_{|U|}\}$, and a set of LRs, $L = \{l_1, l_2, \ldots, l_{|L|}\}$, HASLR builds the backbone graph BBG as follows. First, UCs are aligned against LRs. Each alignment can be encoded by a 7-tuple $\big(rbeg,\ rend,\ uid,\ ustrand,\ ubeg,\ uend,\ nmatch\big)$ whose elements respectively denote the start and end positions of the alignment on the LR, the index of the UC in U, the strand of the alignment ($+$ or $-$), the start and end position of the alignment on the UC, and the number of matched bases in the alignment. Let $A_i = \big(a^i_1, a^i_2, \ldots a^i_{|A_i|}\big)$ be the list of alignments of UCs to l_i, sorted by $rend$.

Note that alignments in A_i may overlap due to relaxed alignment parameters in order to account for the high sequencing error rate of LRs. Thus, in the next step we aim to select a subset of non-overlapping alignments whose total identity score – defined as the sum of the number of matched bases – is maximal. Let $S_i(j)$ be the best subset among the first j alignments, i.e. the non-overlapping subset of these j alignments with maximal total identity score. $S_i(j)$ can be calculated using the following dynamic programming formulation:

$$S_i(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\Big\{S_i\big(j-1\big)\,,\, S_i\big(\text{prev}(j)\big) + a^i_j[nmatch]\Big\} & \text{otherwise} \end{cases} \tag{1}$$

where $\text{prev}(j)$ is the largest index $z < j$ such that a^i_j and a^i_z are non-overlapping alignments. By calculating $S_i(|A_i|)$ and backtracking, we obtain a sorted sub-list $R_i = (r^i_1, r^i_2, \ldots, r^i_{|R_i|})$ of non-overlapping alignments with maximal total identity score, which we call the *compact representation* of read l_i. Note that since the input list is sorted, $\text{prev}(.)$ can be calculated in logarithmic time which makes the time complexity of this dynamic programming $O(|A_i| \log |A_i|)$.

The backbone graph is a *directed* graph $BBG = (V, E)$. The set of nodes is defined as $V = \{u^+_j, u^-_j \mid 1 \le j \le |U|\}$ where u^+_j and u^-_j represent the forward and reverse strand of the UC u_j, respectively. The set of edges is defined as the oriented adjacencies between UCs implied by the compact representations of LRs. Formally each edge is represented by a triplet $(u_h, u_t, supp)$ where $u_h, u_t \in V$ and $supp$ is the set of indices of LRs supporting the adjacency between u_h and u_t; these triplets are obtained as follows:

$$E = \bigcup_{1 \le i \le |L|\,,\, 1 \le j < |R_i|} \Big\{ \Big(u^{hs}_h, u^{ts}_t, \{i\}\Big), \Big(u^{REV(ts)}_t, u^{REV(hs)}_h, \{i\}\Big) \Big\}$$

where $h = r^i_j[uid]$, $hs = r^i_j[ustrand]$, $t = r^i_{j+1}[uid]$, $ts = r^i_{j+1}[ustrand]$, $REV(+) = -$, and $REV(-) = +$. Supplemental Figure S11 illustrates the construction of the backbone graph edges for several combinations of UC alignments on LRs.

At the end of this stage, the resulting backbone graph is a multi-graph as there can be multiple edges between two nodes with different *supp*. In order to make it easier to process the backbone graph, we convert it into a simple graph by merging *supp* of all edges between every pair of nodes into a set of supporting LRs.

S3.3 Graph cleaning and simplification

Ideally, with accurate identification of UCs and correct alignment of UCs onto LRs, the backbone graph for a *haploid genome* will consist of a set of connected components, each of which is a *simple path* of nodes. In practice, this ideal case does not happen – mainly due to sequencing errors, wrong UC to LR alignments, and chimeric reads. As a result, some fake branches as well as artifactual structures might be formed in the backbone graph.

We clean the backbone graph BBG in two stages. First, in order to reduce the effect of wrong UC to LR alignments, we remove all edges e such that $|e[supp]| < minSupp$, for a given parameter $minSupp$. Second, the graph is simplified to remove the artifactual structures. These structures are known as *tips* and *bubbles*. Tips are dead-end simple paths whose length are small compared to their *parallel* paths. Bubbles are formed when two disjoint simple paths occur between two nodes. Supplemental Figure S12 shows examples of tips and bubbles in our backbone graph. There exist well-known algorithms for removing tips and bubbles that are commonly used in assemblers (Zerbino and Birney, 2008; Bankevich et al., 2012; Molnar et al., 2017). We adapt these algorithms for use in HASLR. Note that our tip and bubble removal procedures require an estimation of the length of simple paths. Such estimation can be obtained from the length of UCs corresponding to the nodes contained in a simple path as well as the average length of all LR subsequences that are supporting edges between consecutive nodes. In the following we provide more details about our tip and bubble removal steps.

Estimation of length and coverage for simple paths. In order to perform tip and bubble removal, HASLR requires an estimate for the length and coverage of each simple path. Here, we explain how this estimation is calculated.

For each UC in a simple path, we can calculate the coordinates of region that is aligned to all long reads (we refer to this region as *shared* region). Since the length of shared regions corresponding to all UCs are known, we only need to find an estimation for the middle regions (between two consecutive shared regions). To do this, for each long read supporting the edge connecting two UCs, we calculate the length of the LR subsequence that falls between shared regions (using the alignment's CIGAR string). See Supplemental Figure S13 for a toy example. We use the average of length of all these subsequences as the estimation for the region between shared regions. Finally, the length of the simple path can be estimated as the sum of length of all shared regions plus the estimated length of all middle regions.

In addition, the coverage of each simple path can be calculated based on the number of long reads supporting each edge as well as the estimated length of the middle regions between two consecutive shared regions.

Bubble removal. On a haploid genome, our identification of unique short read contigs is accurate, bubbles are caused only by incorrect alignment of UCs in the middle of LRs. In this case, the bubble is usually formed by two simple paths with same length while one of them has a significantly lower coverage.

In contrast, in diploid genomes, it is possible to have natural bubbles corresponding to heterozygous regions of the genome. The main characteristic of such bubbles is having similar coverage on two paths forming the bubble. If the region contains a heterozygous insertion or deletion, the length of two simple paths forming the bubble are different. On the other hand, if the region contains an inversion, two paths have the same length. Therefore, looking at length of the two paths forming the bubble is not a good criteria for identification of artificial bubbles. This means, decision making should be solely based on the coverage of two paths.

Tip removal. Tips are mainly caused by incorrect alignment of UCs at the extremities of LRs. As a result, the simple path causing the tip is expected to have a small length. In addition, the coverage of such simple path is usually much lower than other simple paths. In our implementation, a simple path is considered as tip if (i) it is a dead-end (only one end is connected to other nodes) and (ii) contains less than 3 UCs. Based on our observations, most of the tips are dead-end simple paths that contain only a single UC.

S3.4 Generating the assembly

Let G be the cleaned and simplified backbone graph. The principle behind the construction of the assembly is that each simple path in the cleaned backbone graph G is used to define a contig of this assembly. Suppose $P = (v_1, e_{12}, v_2, e_{23}, v_3, \ldots, v_n)$ is a simple path of G. Although we already have the DNA sequence for each UC corresponding to each node v_i, the DNA sequence of the resulting contig cannot be obtained immediately. This is due to the fact that at this stage the subsequence between v_i and v_{i+1} is unknown for each $1 \leq i < n$. Here, we explain how these missing subsequences are reconstructed.

For simplicity, suppose we would like to obtain the subsequence between the pair v_1 and v_2 in P. Note that by construction, $e_{12}[supp]$ contains all LRs supporting e_{12}. We can extract the subsequence between v_1 and v_2 from each LR in $e_{12}[supp]$. To do this, we find the region of UCs corresponding to v_1 and v_2 that are aligned to all LRs in $e_{12}[supp]$. Using the alignment transcript (i.e. CIGAR string) the unaligned coordinate of each long read is calculated (see Supplemental Figure S13 for a toy example). By computing the consensus sequence of the extracted subsequences, we obtain cns_{12}. Therefore, the DNA sequence corresponding to P can be obtained via $CONCAT(u_1, cns_{12}, u_2, cns_{23}, u_3, \ldots, u_n)$ where $CONCAT(.)$ returns the concatenated DNA sequence of all its arguments.

In order to generate the assembly, HASLR extracts all the simple paths in the cleaned backbone graph G and constructs the corresponding contig for each of them as explained above. It is important to note that each simple path P has a *twin* path P' which corresponds to the reverse complement of the contig generated from P. Therefore, during our simple path extraction procedure, we ensure to not use twin paths to avoid redundancy.

S3.5 Implementation details.

(i) HASLR utilizes a SR assembler to build its initial SR contigs. However, a higher quality assembly that has fewer misassemblies is preferred. For this purpose, HASLR utilizes Minia (Chikhi and Rizk, 2013) to assemble SRs into SR contigs. Based on our experiments, Minia can generate a high quality assembly quickly using a small memory footprint. (ii) For finding UCs, HASLR calculates mean k-mer frequencies with a small value of k (default $k = 49$). This information can be easily obtained by performing a k-mer counting on the SR dataset (for example using KMC (Kokot et al., 2017)) and calculating the average k-mer count of all k-mers present in each SR contig. Nevertheless, usually assemblers automatically provide such information (e.g Minia and SPAdes). HASLR takes k-mer frequencies reported by Minia for this task. (iii) HASLR uses only longest $25\times$ coverage of long reads for building the backbone graph which are extracted based on the given expected genome size. (iv) In order to align UCs to LRs, HASLR employs minimap2 (Li, 2018). (v) Graph cleaning is done with $minSupp = 3$ meaning that any edge that is supported with less than 3 LRs is discarded. (vi) Finally, consensus sequences are obtained using the Partial Order Alignment (Lee et al., 2002; Lee, 2003) (POA) algorithm implemented in the SPOA package (Vaser et al., 2017). We have provided the versions of the tools and the parameters that are used to execute them in Supplemental Table S1 and Supplemental Section S5, respectively.

S4 Simulated data

We used PBSIM to generate the simulated datasets. PBSIM has an option to infer the mean and standard deviation of read length and the error rate from a real dataset. So first, we prepare that real dataset. We use the first 10 runs of CHM1 (P6C4) dataset:

```
$ for acc in SRR2183739 SRR2183740 SRR2183741 SRR2183742 SRR2183743 SRR2183744 SRR2183745
    SRR2183746 SRR2183747 SRR2183748; do wget http://sra-download.ncbi.nlm.nih.gov/srapub_files/${
    acc}_${acc}_hdf5.tgz; done

$ for acc in SRR2183739 SRR2183740 SRR2183741 SRR2183742 SRR2183743 SRR2183744 SRR2183745
    SRR2183746 SRR2183747 SRR2183748; do tar -zxvf ${acc}_${acc}_hdf5.tgz; done

$ for bax in m15051*.bax.h5; do bash5tools.py ${bax} --outFilePrefix ${bax} --outType fastq --
    readType subreads --minLength 50 --minReadScore 0.75; done

$ for seq in m15051*.fastq; do cat ${seq}; done > chm1_p6c4_first_10.fastq
```

For simulation of the long reads:

```
$ pbsim --seed 0 --data-type CLR --depth 50 --length-min 1 --length-max 500000 --sample-fastq
    chm1_p6c4_first_10.fastq --prefix long <reference_fasta>
```

For simulation of the short reads:

```
$ art_illumina --paired --in <reference_fasta> --len 150 --mflen 500 --sdev 50 --fcov 50 --rndSeed
    0 --noALN --out short
```

S5 Command details

- Running HASLR

```
$ python3 haslr.py --threads <cores> --type <pacbio|nanopore> --cov-lr 25 --minia-kmer 55 --
    minia-solid 3 --aln-block 500 --out <output_directory> --genome <genome_size> --long <
    lr_file> --short <sr_file_1> <sr_file_2>
```

- Running Canu

```
$ canu -p <assembly_prefix> -d <output_directory> genomeSize=<genome_size> -pacbio-raw <
    lr_file> useGrid=false
```

- Running Flye

```
$ flye -t <cores> -o <output_directory> -g <genome_size> --pacbio-raw <lr_file>
```

- Running wtdbg2

```
$ perl wtdbg2.pl -t <cores> -x <rs|ont> -g <genome_size> -o <assembly_prefix> <lr_file>
```

- Running miniasm

```
$ minimap2 -t <cores> -x ava-pb <lr_file> <lr_file> > asm.ava.paf

$ miniasm -f <lr_file> asm.ava.paf > asm.graph.gfa

$ awk '/^S/{print ">"$2"\n"$3}' asm.graph.gfa > asm.draft.fa

$ minimap2 -t <cores> -x map-pb asm.draft.fa <lr_file> > asm.map.paf

$ racon -t <cores> <lr_file> asm.map.paf asm.draft.fa > asm.polish.fa
```

- Running SPAdes

```
$ spades.py -t <cores> -m <max_memory> -1 <sr_file_1> -2 <sr_file_2> -o <output_directory>
```

- Running hybridSPAdes

```
$ spades.py -t <cores> -m <max_memory> -1 <sr_file_1> -2 <sr_file_2> --pacbio <lr_file> -o <
    output_directory>
```

- Running Unicycler

```
$ unicycler -t <cores> --no_rotate --no_miniasm --no_pilon -o <assembly_prefix> -1 <
    sr_file_1> -2 <sr_file_2> -l <lr_file>
```

- Running DBG2OLC (based on suggestions on the github repository)

```
$ fastutils interleave -q -1 <sr_file_1> -2 <sr_file_2> | fastutils subsample -q -d 50 -g <
    genome_size> > short.50x.fastq

$ fastutils subsample -l -d 30 -g <genome_size> -i <lr_file> > long.30x.fasta

$ SparseAssembler LD 0 k 51 g 15 NodeCovTh 1 EdgeCovTh 0 GS <genome_size> f short.50x.fastq

$ DBG2OLC k 17 AdaptiveTh 0.01 KmerCovTh 2 MinOverlap 20 RemoveChimera 1 Contigs Contigs.txt
     f long.30x.fasta

$ cat Contigs.txt long.30x.fasta > ctg_pb.fasta

$ ulimit -n 4000

$ split_and_run_sparc.sh backbone_raw.fasta DBG2OLC_Consensus_info.txt ctg_pb.fasta ./
    consensus_dir
```

- Running MaSuRCA
  Content of config.txt

```
DATA
PE= pe <insert_mean> <insert_std> <sr_file_1> <sr_file_1>
PACBIO=<lr_file>
# NANOPORE=<lr_file>
END

PARAMETERS
GRAPH_KMER_SIZE = auto
LHE_COVERAGE=25
CA_PARAMETERS = cgwErrorRate=0.15
KMER_COUNT_THRESHOLD = 1
CLOSE_GAPS=0
NUM_THREADS = <cores>
JF_SIZE = 200000000
END
```

  Command

```
   bash assemble.sh
```

- Running Wengan

```
   perl wengan.pl -t <cores> -a M -p <assembly_prefix> -x <pacraw|ontraw> -g <genome_size> -s <
       sr_file_1>,<sr_file_1> -l <lr_file>
```

Supplemental References

D. Antipov, A. Korobeynikov, J. S. McLean, and P. A. Pevzner. hybridspades: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32(7):1009–1015, 2015.

A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

R. Chikhi and G. Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):22, 2013.

A. Di Genova, E. Buena-Atienza, S. Ossowski, and M.-F. Sagot. Wengan: Efficient and high quality hybrid de novo assembly of human genomes. *bioRxiv*, page 840447, 2019.

W. Huang, L. Li, J. R. Myers, and G. T. Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2011.

M. Kokot, M. Długosz, and S. Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540–546, 2019.

S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

C. Lee. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics*, 19(8):999–1008, 2003.

C. Lee, C. Grasso, and M. F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

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

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

A. Mikheenko, A. Prjibelski, V. Saveliev, D. Antipov, and A. Gurevich. Versatile genome assembly evaluation with quast-lg. *Bioinformatics*, 34(13):i142–i150, 2018.

M. Molnar, E. Haghshenas, and L. Ilie. Sage2: parallel human genome assembly. *Bioinformatics*, 34(4):678–680, 2017.

Y. Ono, K. Asai, and M. Hamada. Pbsim: Pacbio reads simulatortoward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2012.

J. Ruan and H. Li. Fast and accurate long-read assembly with wtdbg2. *BioRxiv*, page 530972, 2019.

F. A. Simão, R. M. Waterhouse, P. Ioannidis, E. V. Kriventseva, and E. M. Zdobnov. Busco: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212, 2015.

J. T. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, 22(3):549–556, 2012.

J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

R. Vaser, I. Sović, N. Nagarajan, and M. Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017.

R. R. Wick, M. B. Schultz, J. Zobel, and K. E. Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20):3350–3352, 2015.

R. R. Wick, L. M. Judd, C. L. Gorrie, and K. E. Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS computational biology*, 13(6):e1005595, 2017.

C. Ye, C. M. Hill, S. Wu, J. Ruan, and Z. S. Ma. Dbg2olc: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. *Scientific reports*, 6:31900, 2016.

D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

A. V. Zimin, D. Puiu, M.-C. Luo, T. Zhu, S. Koren, G. Marçais, J. A. Yorke, J. Dvořák, and S. L. Salzberg. Hybrid assembly of the large and highly repetitive genome of aegilops tauschii, a progenitor of bread wheat, with the masurca mega-reads algorithm. *Genome research*, 27(5):787–792, 2017.