CONSENT: Consensus-based Self-Correction of Third Generation Sequencing Data

Pierre Morisse [1], Camille Marchet [2], Antoine Limasset [2],
Arnaud Lefebvre [1], Thierry Lecroq [1]

[1] Normandie Univ, UNIROUEN, LITIS, Rouen 76000, France.
[2] Lille Univ, CNRS, Inria, CRIStAL, Lille 59000, France.

DSB 2019
February 6, 2019

Introduction

Context

- 2011: Inception of third generation sequencing technologies
- Two main technologies: Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT)
- Sequencing of much longer reads, tens of kbps on average, up to 1Mb (ONT ultra-long reads)
- Expected to solve various problem in the genome assembly field

Introduction

Context

- Long reads (LR) are very noisy (10-30% error rate)
- Display complex error profiles (errors are mostly indels)
- Efficient error correction is mandatory
- Two main approaches: hybrid correction and self-correction

Introduction

Hybrid correction

- First efficient approach for LR error correction
- Makes use of complementary short reads (SR) data
- Different approaches: Alignment of SRs to the LRs, use of a De Bruijn graph (DBG), ...
- Particularly useful on old sequencing experiments (very high error rates)

Introduction

Self-correction

- Corrects the LRs solely based on the information they contain
- Third generation sequencing technologies evolve fast
- Error rates of the LRs now reach 10-12% on average
- Error correction still needed
- Self-correction is now a viable alternative

Introduction

Self-correction

State-of-the-art: Two main approaches

1. Compute overlaps between the LRs
2. Build a DBG from solid *k*-mers of the LRs (LoRMA [Salmela et al., 2016])

Introduction

Self-correction

- Overlapping can be performed via:
  - Mapping (Canu [Koren et al., 2017], MECAT [Xiao et al., 2017], FLAS [Bao et al., 2018])
  - Alignment (PBDAGCon [Chin et al., 2013], daccord [Tischler and Myers, 2017])
- Two main approaches are then used

Introduction

Multiple alignment

- Build a directed acyclic graph (DAG) to represent the alignments and compute consensus

```
ACCAAGGT   R1
ACAAGGGT   R2

ACCAAGGT   R1
ACCAA..T   R3
```


De Bruijn graph

- Divide the alignments into small windows
- Correct the windows independently with DBGs

```
.GATCGGG..TAT.TGCCCGTGTTTATGCGTGTG   R1
TGTTCAGGCAAATATG...GAAACAAGGCCTG..   R2

GAT..CGGGTATTGCCCGTGTTTATGCGTG..TG   R1
TATTTCTG..AT.GCGC.TGACTTTTCTTGGCAG   R3
```

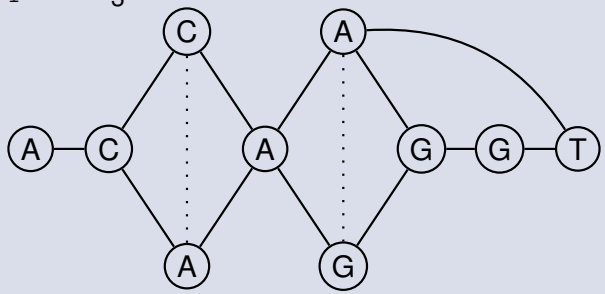
Introduction

Multiple alignment

- Build a directed acyclic graph (DAG) to represent the alignments and compute consensus

```
ACCAAGGT    R1
ACAAGGGT    R2

ACCAAGGT    R1
ACCAA..T    R3
```



De Bruijn graph

- Divide the alignments into small windows
- Correct the windows independently with DBGs

```
.GATCGGG..TAT.TGCCCGTGTTTATGCGTGTG    R1
TGTTCAGGCAAATATG...GAAACAAGGCCTG..    R2

GAT..CGGGTATTGCCCGTGTTTATGCGTG..TG    R1
TATTTCTG..AT.GCGC.TGACTTTTCTTGGCAG    R3
```

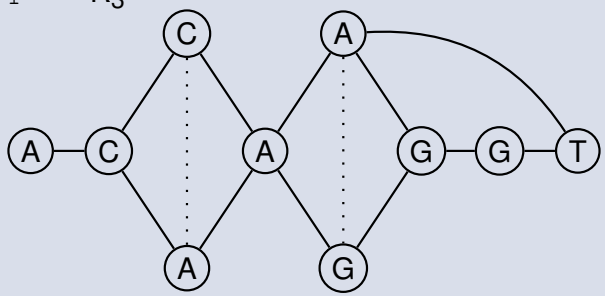
Introduction

Multiple alignment

- Build a directed acyclic graph (DAG) to represent the alignments and compute consensus

```
ACCAAGGT    R1
ACAAGGGT    R2

ACCAAGGT    R1
ACCAA..T    R3
```



De Bruijn graph

- Divide the alignments into small windows
- Correct the windows independently with DBGs

```
.GATCGGG..TAT.TGCCCGTGTTTATGCGTGTG     R1
TGTTCAGGCAAATATG...GAAACAAGGCCTG..     R2

GAT..CGGGTATTGCCCGTGTTTATGCGTG..TG     R1
TATTTCTG..AT.GCGC.TGACTTTTCTTGGCAG     R3
```

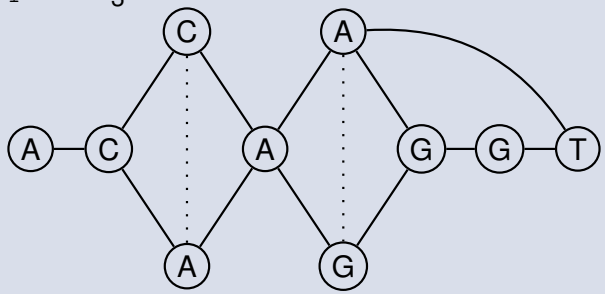
Introduction

Multiple alignment

- Build a directed acyclic graph (DAG) to represent the alignments and compute consensus

```
ACCAAGGT    R1
ACAAGGGT    R2

ACCAAGGT    R1
ACCAA..T    R3
```



De Bruijn graph

- Divide the alignments into small windows
- Correct the windows independently with DBGs

```
.GATCGGG..TAT.TGCCCGTGTTTATGCGTGTG     R1
TGTTCAGGCAAATATG...GAAACAAGGCCTG..     R2

GAT..CGGGTATTGCCCGTGTTTATGCGTG..TG     R1
TATTTCTG..AT.GCGC.TGACTTTTCTTGGCAG     R3
```

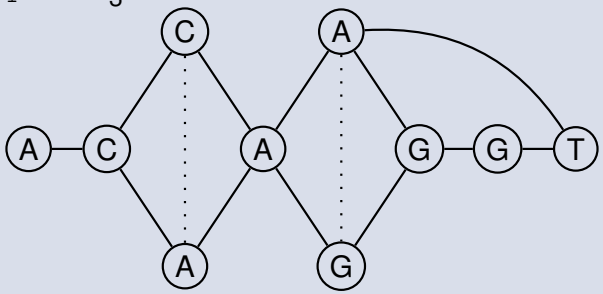
Introduction

Multiple alignment

- Build a directed acyclic graph (DAG) to represent the alignments and compute consensus

```
ACCAAGGT    R1
ACAAGGGT    R2

ACCAAGGT    R1
ACCAA..T    R3
```



De Bruijn graph

- Divide the alignments into small windows
- Correct the windows independently with DBGs

```
.GATCGGG..TAT.TGCCCGTGTTTATGCGTGTG    R1
TGTTCAGGCAAATATG...GAAACAAGGCCTG..    R2

GAT..CGGGTATTGCCCGTGTTTATGCGTG..TG    R1
TATTTCTG..AT.GCGC.TGACTTTTCTTGGCAG    R3
```

Introduction

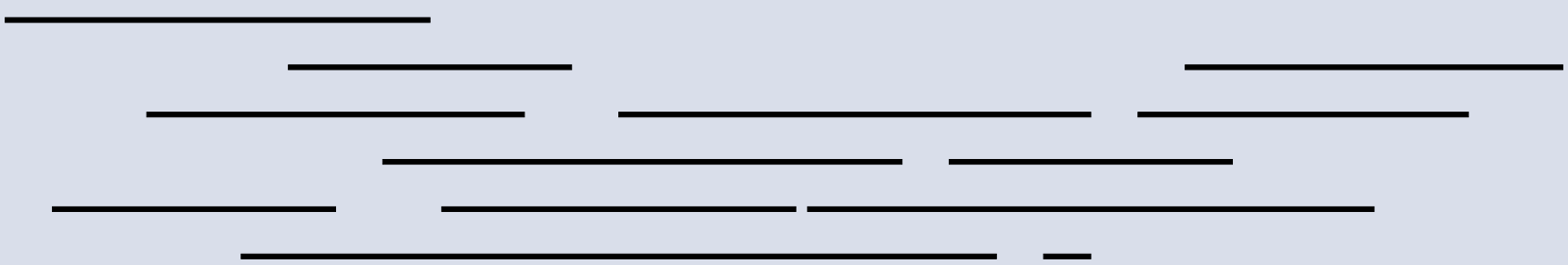
Contribution

- We introduce CONSENT, a new self-correction method combining both previous strategies:
- Alignments are divided into windows
- Windows consensus are computed using DAGs
- Windows consensus are polished with the help of local DBGs
- Compared to SOTA: Comparable results, better scalability

Pre-treatment

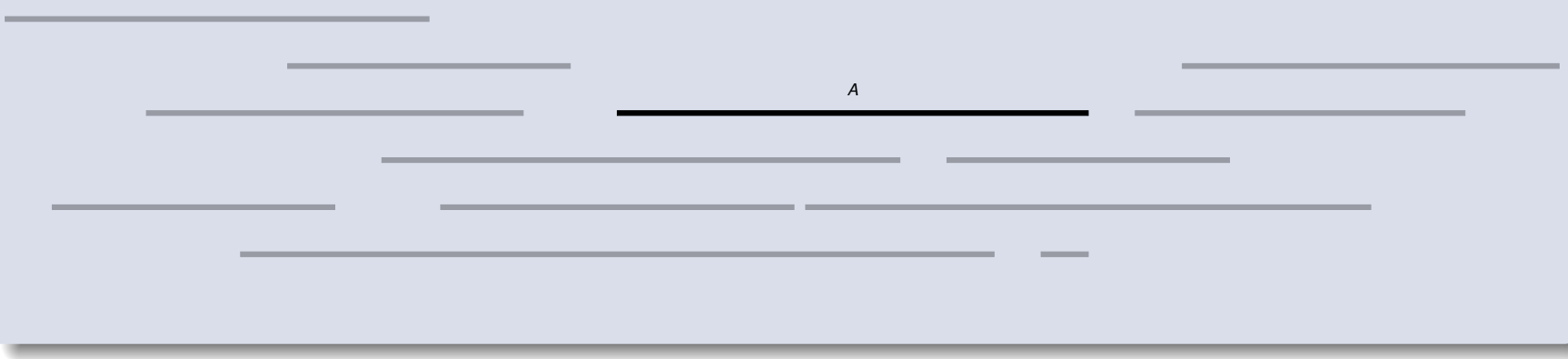
Overlap the long reads

Via mapping, with Minimap2 [Li, 2018]



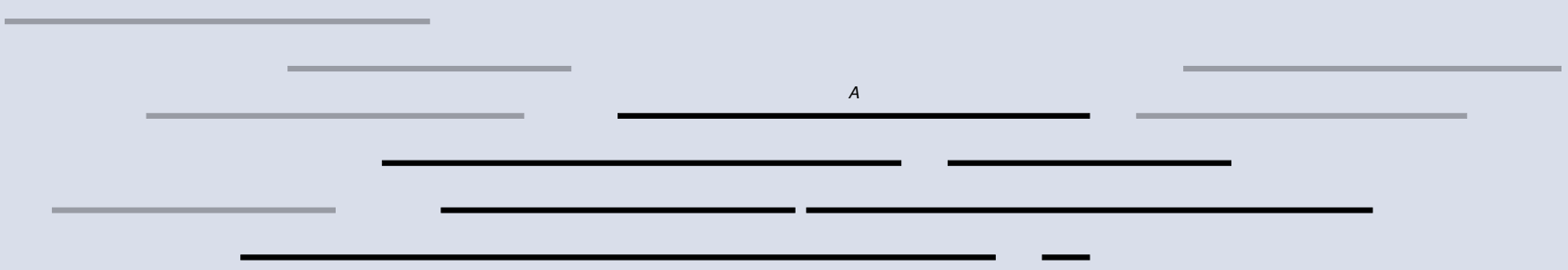
First step: Retrieve alignment pile

Select a long read to correct



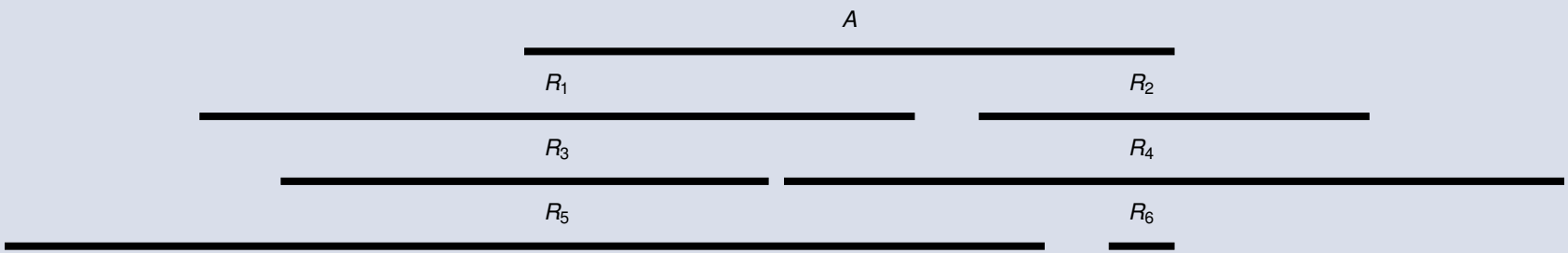
First step: Retrieve alignment pile

Retrieve overlapping long reads



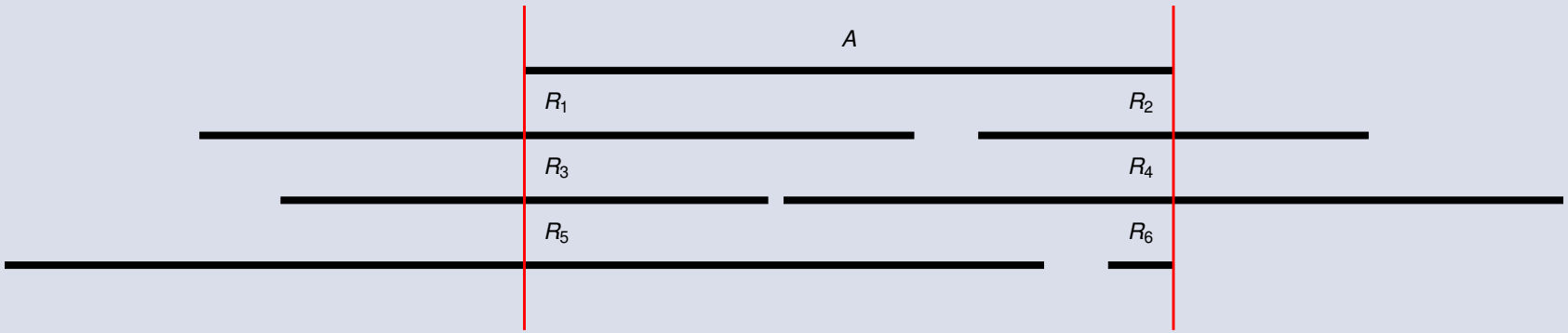
First step: Retrieve alignment pile

Get the alignment pile



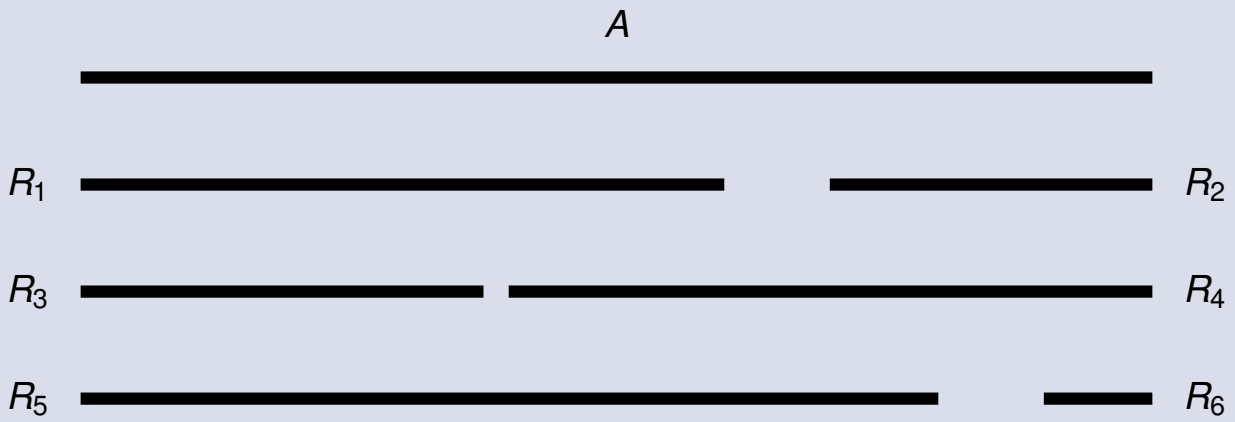
First step: Retrieve alignment pile

Trim the alignment pile



First step: Retrieve alignment piles

Trim the alignment pile



Second step: Divide piles into windows

Definition

A window $w = (beg, end)$ is a "factor" of an alignment pile

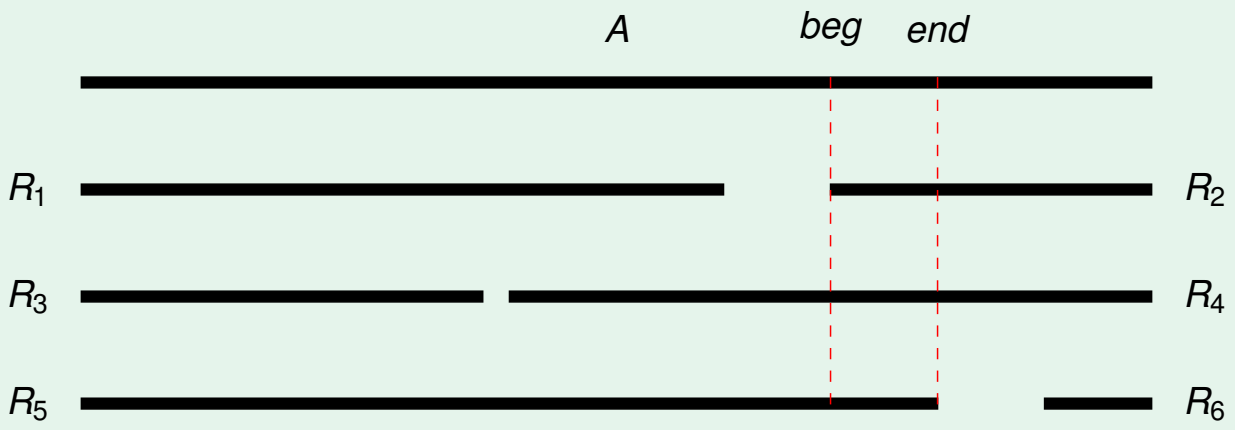
Example

Second step: Divide piles into windows

Definition

A window $w = (beg, end)$ is a "factor" of an alignment pile

Example



Second step: Divide piles into windows

For correction, we will only consider windows $w = (beg, end)$ such as:

- $end - beg + 1 = l$
- $\forall i, beg \leq i \leq end$, i is covered by at least c reads

Example

On the previous example, with $c = 4$:

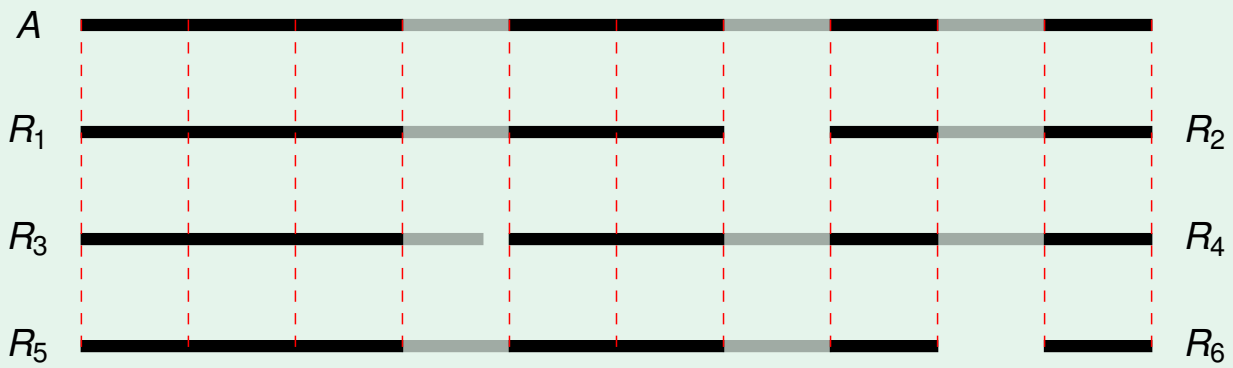
Second step: Divide piles into windows

For correction, we will only consider windows $w = (beg, end)$ such as:

- $end - beg + 1 = l$
- $\forall i, beg \leq i \leq end$, i is covered by at least c reads

Example

On the previous example, with $c = 4$:



Third step: Compute consensus of a window

2. Compute consensus

- Compute multiple sequence alignment (MSA) of these sequences
- Compute consensus from the MSA
- $\Rightarrow$ POA [Lee et al., 2002]

Third step: Compute consensus of a window

POA (Partial Order Alignment)

- Multiple sequence alignment strategy based on partial order graphs
- Two interests:
  1. Computes *actual* multiple sequence alignment
  2. Directly builds the DAG representing the multiple alignment

Third step: Compute consensus of a window

POA (Partial Order Alignment)

- Multiple sequence alignment strategy based on partial order graphs
- Two interests:
  1. Computes *actual* multiple sequence alignment
  2. Directly builds the DAG representing the multiple alignment

Third step: Compute consensus of a window

POA (Partial Order Alignment)

- Multiple sequence alignment strategy based on partial order graphs
- Two interests:
  1. Computes *actual* multiple sequence alignment
  2. Directly builds the DAG representing the multiple alignment

Third step: Compute consensus of a window

POA

Workflow:

- Start with a graph only containing the first sequence
- Insert new sequences with a generalization of the Needleman-Wunsch algorithm

Third step: Compute consensus of a window

Segmentation strategy

- In practice, we use windows of a few hundred bases
- POA is time consuming
- We developed a segmentation strategy
- Compute MSA and consensus for smaller sequences $\Rightarrow$ faster

Third step: Compute consensus of a window

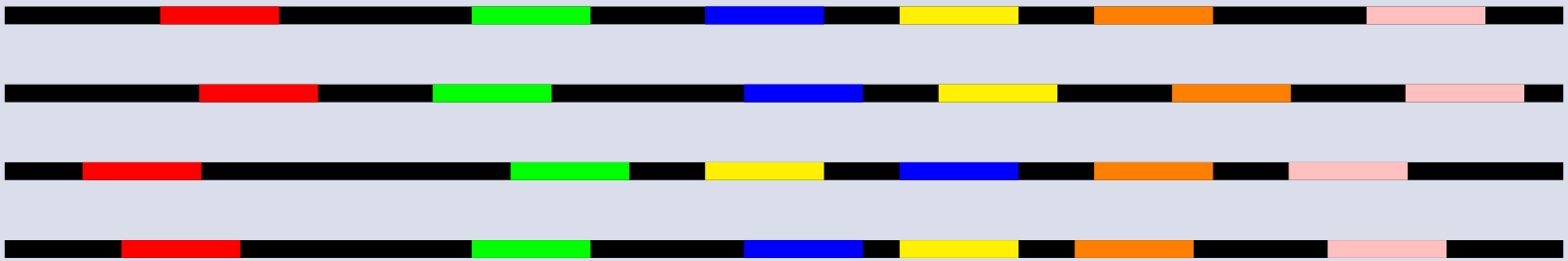
Segmentation strategy

1. Compute shared anchors between the window's sequences

Third step: Compute consensus of a window

Segmentation strategy

1. Compute shared anchors between the window's sequences



Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

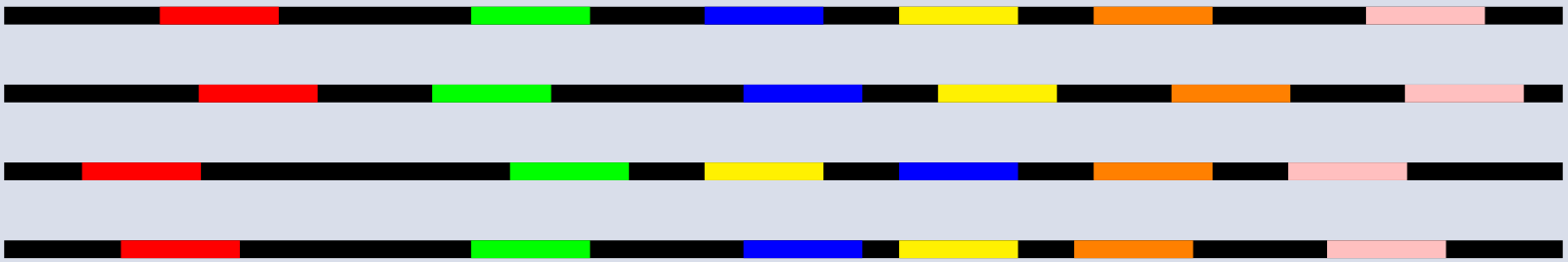
1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i



Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

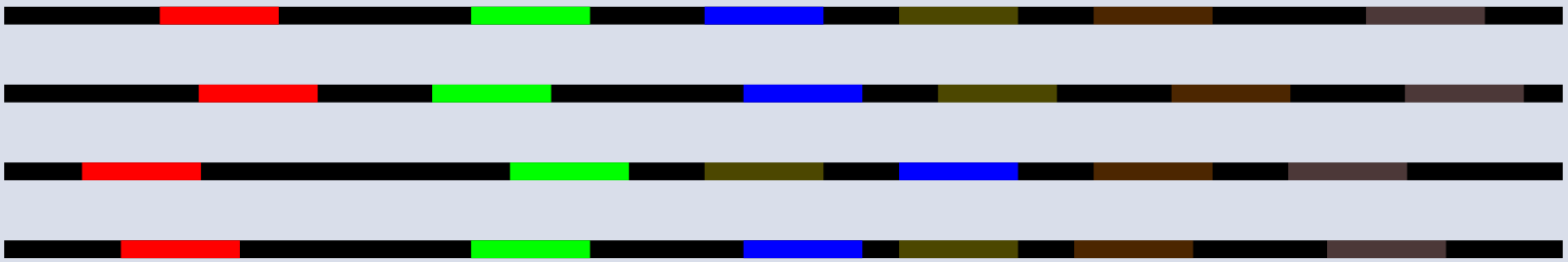
1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

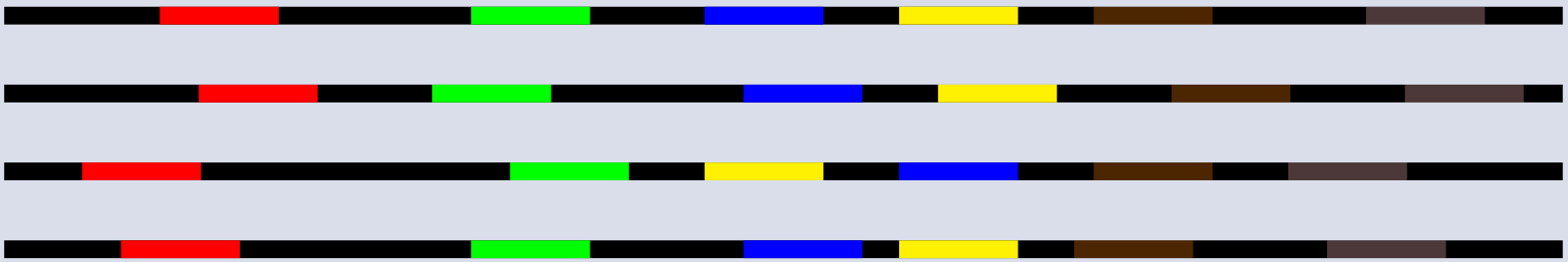


Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

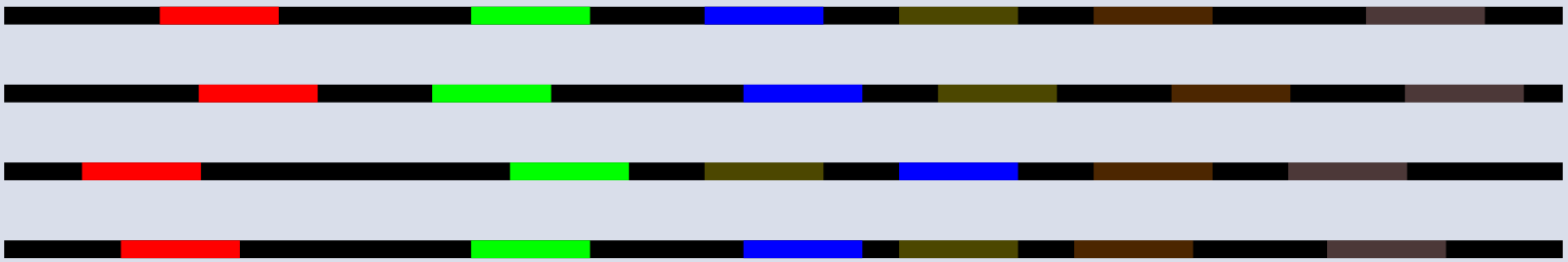


Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

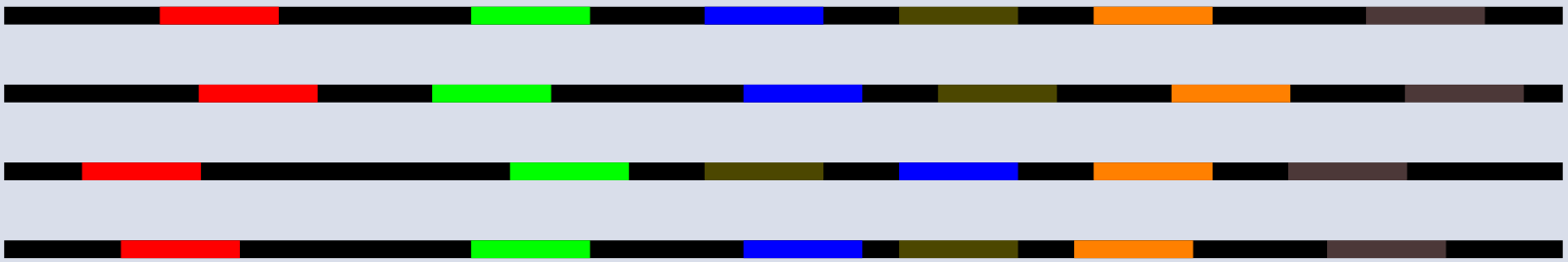


Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i

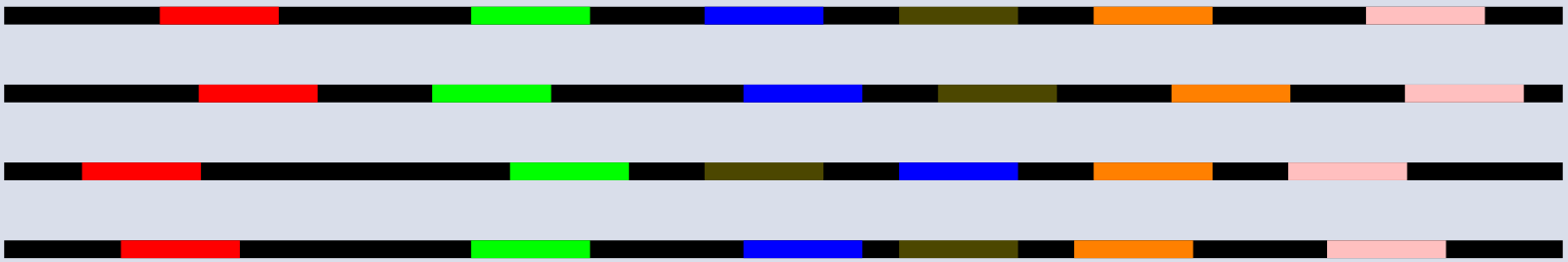


Third step: Compute consensus of a window

Segmentation strategy

2. Search for the longest anchors chain such as $\forall A_i, A_{i+1}$:

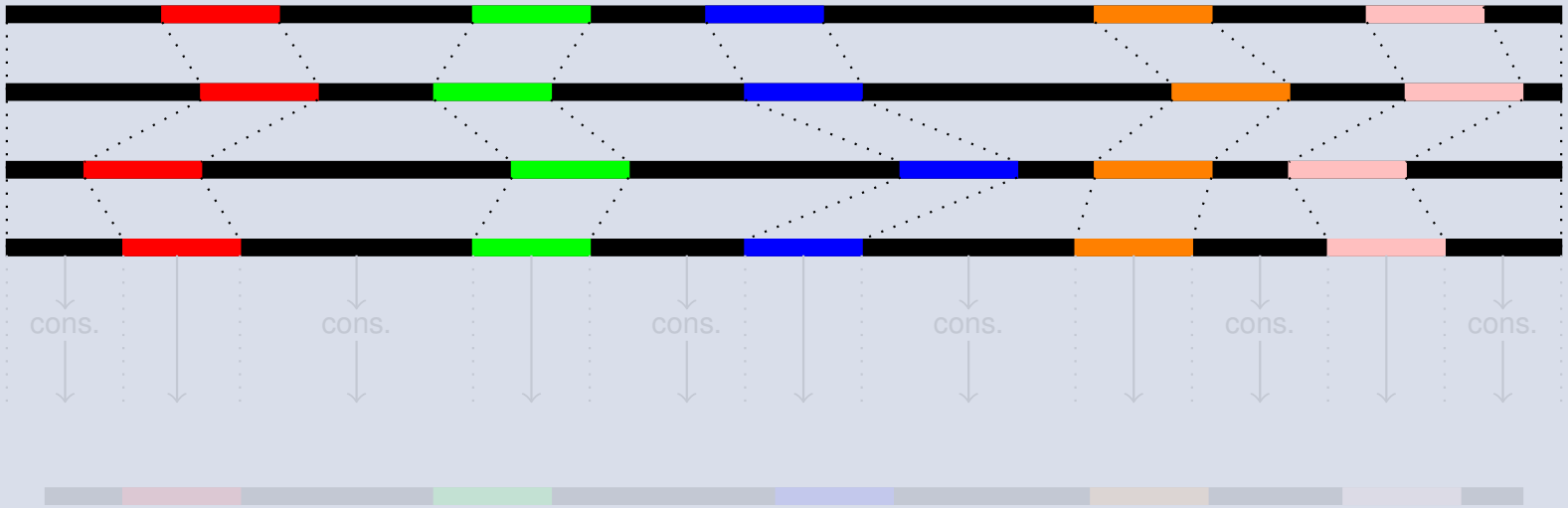
1. A_i is followed by A_{i+1} in at least N sequences
2. A_{i+1} is never followed by A_i



Third step: Compute consensus of a window

Segmentation strategy

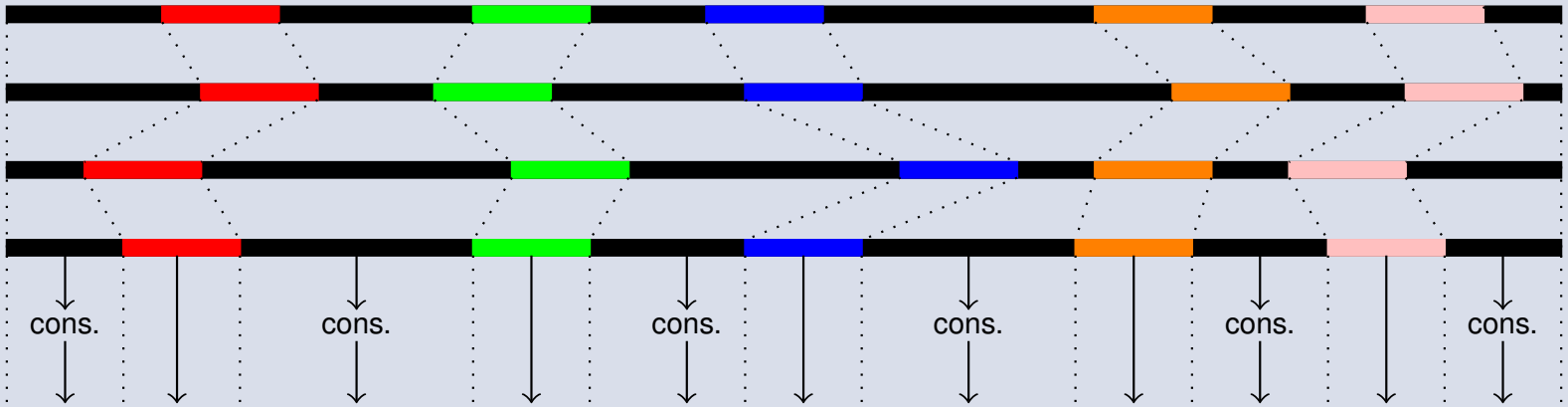
3. Compute MSA / consensus for sequences bordered by anchors



Third step: Compute consensus of a window

Segmentation strategy

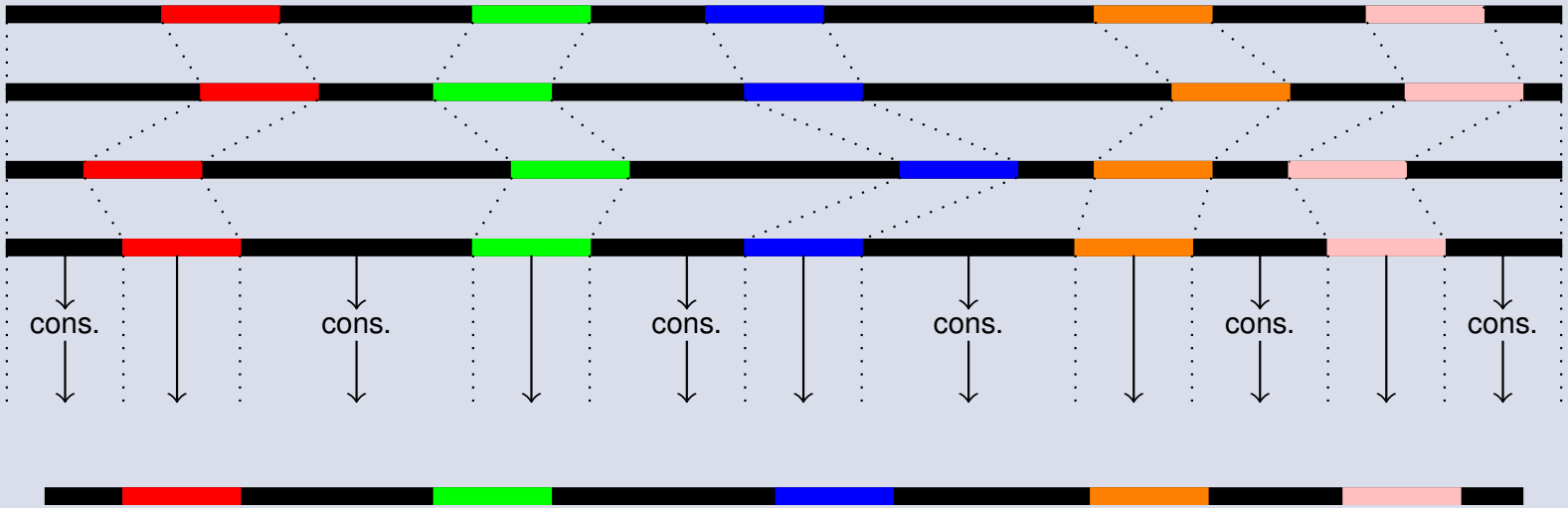
3. Compute MSA / consensus for sequences bordered by anchors



Third step: Compute consensus of a window

Segmentation strategy

3. Compute MSA / consensus for sequences bordered by anchors



Fourth step: Polish the consensus

Approach

- Build a DBG from the window's sequences
- Consensus ⇒ solid *k*-mers in uppercase, weak *k*-mers in lowercase

```
GATCGGGTcatTGCCCGTGTTTATGCGTGtg
```

- Correct lowercase regions
- Bordered regions ⇒ Traverse the graph to find a path between solid, anchor *k*-mers
- Extremities ⇒ Traverse the graph as much as possible

Fifth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the polished consensus
- Locally align it to the LR, around the positions of the window
- Aligned factor of the LR replaced by aligned factor of the consensus
- Repeat with other windows (in practice, overlapping)

Segmentation strategy validation

Results

Simulated PacBio dataset from *E. coli*, 50x, 12% error rate

	Without segmentation	With segmentation
Throughput	214,667,382	215,693,736
Error rate (%)	0.0757	0.0722
Runtime	5h31min	7min
Memory (MB)	750	675

Comparison to state-of-the-art

Datasets

Dataset	Number of reads	Average length	Error rate	Coverage
Simulated Pacific Biosciences data				
E. coli 30x	16,959	8,235	12.29	30x
E. coli 60x	33,918	8,211	12.28	60x
S. cerevisiae 30x	45,198	8,216	12.28	30x
S. cerevisiae 60x	90,397	8,204	12.29	60x
C. elegans 30x	366,416	8,204	12.28	30x
C. elegans 60x	732,832	8,220	12.28	60x
Real Oxford Nanopore data				
D. melanogaster	1,327,569	6,828	14.57	63x
H. sapiens, chr1	1,075,867	6,744	17.60	29x

Comparison to state-of-the-art

Compared tools

- Canu
- Daccord
- FLAS
- LoRMA
- MECAT

Comparison to state-of-the-art

Simulated data, 30x coverage

Dataset	Corrector	Throughput (Mbp)	Error rate (%)	Deletions (%)	Insertions (%)	Substitutions (%)	Total Runtime	Total Memory (MB)
E. coli 30x	Original	140	12.2862	2.6447	8.7973	0.8442	N/A	N/A
	Canu	130	0.2508	0.0636	0.2001	0.0102	19min	4,613
	daccord	**131**	**0.0219**	**0.0034**	**0.0090**	0.0115	14 min	6,813
	FLAS	130	0.2077	0.1490	0.0741	0.0043	12min	1,639
	LoRMA	13	0.2969	0.0429	0.1466	0.1322	10min	32,155
	MECAT	107	0.1649	0.1328	0.0459	**0.0018**	**1 min 39 sec**	**1,600**
	CONSENT	130	0.2013	0.0944	0.1095	0.0163	17 min 10 sec	2,390
S. cerevisiae 30x	Original	371	12.283	2.646	8.7937	0.8434	N/A	N/A
	Canu	227	0.8472	0.2335	0.6393	0.0479	29min	3,681
	daccord	**348**	**0.1186**	**0.0222**	**0.0368**	0.0707	1 h 19 min	31,798
	FLAS	345	0.2537	0.1863	0.0828	0.0088	29min	2,935
	LoRMA	52	0.4954	0.0798	0.2690	0.1887	46min	31,899
	MECAT	285	0.2111	0.1691	0.0574	**0.0048**	**5 min**	**2,907**
	CONSENT	345	0.2890	0.1428	0.1386	0.0348	46 min	5,523
C. elegans 30x	Original	3,006	12.2806	2.6449	8.7926	0.8431	N/A	N/A
	Canu	2,776	0.2895	**0.0682**	0.2354	0.0126	9h09min	**6,921**
	daccord	-	-	-	-	-	-	-
	FLAS	2,718	0.3862	0.2656	0.1469	0.0106	3h07min	10,565
	LoRMA	258	1.1573	0.2094	0.4686	0.5764	8h19min	31,827
	MECAT	2,085	**0.2682**	0.2135	**0.0764**	**0.0037**	**48 min**	10,535
	CONSENT	**2,791**	0.6300	0.3064	0.2958	0.0878	9 h 36 min	21,819

Comparison to state-of-the-art

Simulated data, 60x coverage

Dataset	Corrector	Throughput (Mbp)	Error rate (%)	Deletions (%)	Insertions (%)	Substitutions (%)	Total Runtime	Total Memory (MB)
E. coli 60x	Original	279	12.2788	2.6437	8.7919	0.8432	N/A	N/A
	Canu	219	0.5211	0.1390	0.4045	0.0243	24min	3,674
	daccord	**261**	**0.0175**	**0.0026**	**0.0062**	0.0103	54 min	18,450
	FLAS	260	0.1039	0.0907	0.0220	0.0010	38min	2,428
	LoRMA	239	0.0660	0.0098	0.0476	0.0147	1h39min	31,682
	MECAT	233	0.1011	0.0896	0.0203	**0.0008**	**5 min**	**2,387**
	CONSENT	259	0.0590	0,0368	0.0241	0.0037	36 min	4,849
S. cerevisiae 60x	Original	742	12.2886	2.6484	8.7963	0.8439	N/A	N/A
	Canu	600	0.5615	0.1518	0.4309	0.0292	1h11min	**3,710**
	daccord	**696**	**0.0305**	**0.0055**	**0.0180**	0.0100	2 h 26 min	32,190
	FLAS	690	0.1430	0.1215	0.0319	0.0031	1h30min	4,984
	LoRMA	634	0.1160	0.0188	0.0778	0.0301	5h25min	31,828
	MECAT	617	0.1365	0.1189	0.0286	**0.0020**	**16 min**	4,954
	CONSENT	690	0.1418	0.0735	0.0650	0,0166	1 h 46 min	11,325
C. elegans 60x	Original	6,024	12.2825	2.6457	8.7937	0.8432	N/A	N/A
	Canu	5,119	0.6623	_	_	_	9 h 30 min	7,050
	daccord	_	_	_	_	_	_	_
	FLAS	**5,614**	0.2160	_	_	_	10 h 45 min	13,682
	LoRMA	3,388	**0.1446**	_	_	_	31 h 04 min	32,104
	MECAT	4,941	0.1882	_	_	_	**2 h 43 min**	10,563
	CONSENT	5,607	0.4604	_	_	_	27 h 04 min	32,284

Comparison to state-of-the-art

Real data

Dataset	Corrector	Number of reads	Throughput (Mbp)	N50 (bp)	Aligned reads (%)	Alignment identity (%)	Genome coverage (%)	Total Runtime	Total Memory (MB)
D. melanogaster	Original	1,327,569	9,064	11,853	85.52	85.43	98.47	N/A	N/A
	Canu	829,965	6,993	**12,694**	98.05	95.20	97.89	14 h 04 min	10,295
	daccord	–	–	–	–	–	–	–	–
	FLAS	855,275	7,866	11,742	95.65	94.99	98.09	10 h 18 min	18,820
	LoRMA	**1,125,279**	6,386	669	97.05	**98.47**	94.76	23 h 51 min	65,536
	MECAT	849,704	7,288	11,676	**99.87**	96.52	97.34	**1 h 54 min**	**13,443**
	CONSENT	1,065,621	**8,178**	12,297	99.26	96.72	**98.20**	38 h	51,361
H. sapiens	Original	1,075,867	7,256	10,568	88.24	82.40	92.46	N/A	N/A
	Canu	–	–	–	–	–	–	–	–
	daccord	–	–	–	–	–	–	–	–
	FLAS[1]	670,708	5,695	10,198	99.06	91.00	92.37	4 h 57 min	14,957
	LoRMA	737,198	1,247	186	96.50	**97.83**	28.62	13 h 07 min	50,435
	MECAT[1]	667,532	5,479	10,343	**99.95**	91.69	91.44	**1 h 53 min**	**11,075**
	CONSENT	**869,462**	**6,349**	**10,839**	99.59	93.00	**92.40**	8 h 30 min	45,869

[1] ultra-long reads were filtered out

Comparison to state-of-the-art

Contigs polishing

Dataset	Method	Contigs	Aligned contigs	NGA50	NGA75	Genome coverage	Runtime	Memory (MB)
E. coli 60x	Original	1	1	_	_	0.89	N/A	N/A
	RACON	1	1	4,663,914	4,663,914	99.90	2 min	628
	CONSENT	1	1	4,637,588	4,637,588	99.90	7 min	4,192
S. cerevisiae 60x	Original	29	29	_	_	0.87	N/A	N/A
	RACON	29	29	539,433	346,116	96.09	5 min	1,673
	CONSENT	29	29	535,665	334,556	96.12	3 min	9,232
C. elegans 60x	Original	47	46	_	_	0.95	N/A	N/A
	RACON	47	47	5,073,456	2,349,027	99.71	46 min	14,264
	CONSENT	47	47	3,737,577	2,073,591	99.57	1 h 42 min	32,144

Take-home messages

- CONSENT: new long read self-correction method
- Introduces a segmentation strategy allowing fast computation of MSA
- Compares well to the SOTA
- Only method scaling to ONT ultra-long reads
- Available at: `https://github.com/morispi/CONSENT`

Future works

- Optimize the parameters (size of the windows, of k, etc)
- Reduce memory consumption $\Rightarrow$ Split Minimap2 index
- Reduce runtime: Deeply covered windows
  - Computing MSA is expensive
  - Probably repeats $\Rightarrow$ Validation strategy
- Segmentation strategy seems promising $\Rightarrow$ Apply it to a greater scale

The end!

Thanks for your attention!

Questions?

Bao, E., Xie, F., Song, C., and Song, D. (2018).
HALS : Fast and High Throughput Algorithm for.
RECOMB-SEQ 2018, pages 1–7.


Chin, C.-S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E. E., Turner, S. W., and Korlach, J. (2013).
Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data.
Nature Methods, 10:563–569.


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


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

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

Salmela, L., Walve, R., Rivals, E., and Ukkonen, E. (2016).
Accurate selfcorrection of errors in long reads using de Bruijn graphs.
Bioinformatics, page btw321.

Tischler, G. and Myers, E. W. (2017).
Non Hybrid Long Read Consensus Using Local De Bruijn Graph Assembly.
bioRxiv.

Xiao, C. L., Chen, Y., Xie, S. Q., Chen, K. N., Wang, Y., Han, Y., Luo, F., and Xie, Z. (2017).
MECAT: Fast mapping, error correction, and de novo assembly for single-molecule sequencing reads.
Nature Methods, 14(11):1072–1074.