LoRSCo: Long Reads Self-Correction

Pierre Morisse [1], Camille Marchet [2], Antoine Limasset [2], Arnaud Lefebvre [1], Pierre Peterlongo [3], Thierry Lecroq [1]

[1] Normandie Univ, UNIROUEN, LITIS, Rouen 76000, France.
[2] Lille Univ, CNRS, Inria, CRIStAL, Lille 59000, France.
[3] Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France.

SeqBio 2018
November 20, 2018

Introduction

Context

- 2010: Inception of third generation sequencing technologies
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Sequencing of much longer reads, tens of kbps on average, up to 882kb
- 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., 2017])

Introduction

Self-correction

- Overlapping can be performed via:
  - Mapping (Canu [Koren et al., 2017], MECAT [Xiao et al., 2017])
  - 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
```

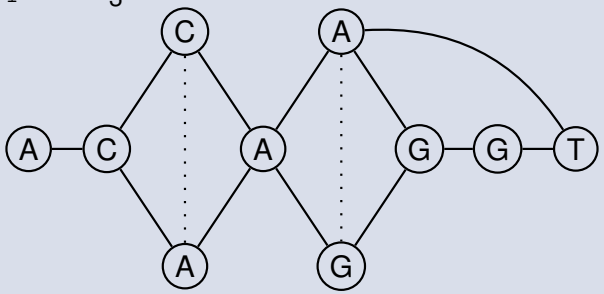
Introduction

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
```

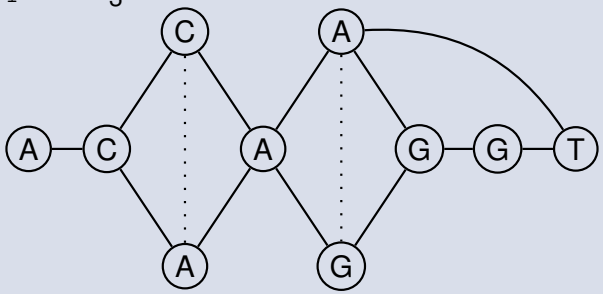
Introduction

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
```

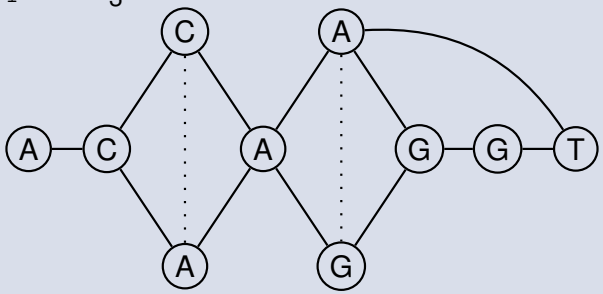
Introduction

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
```

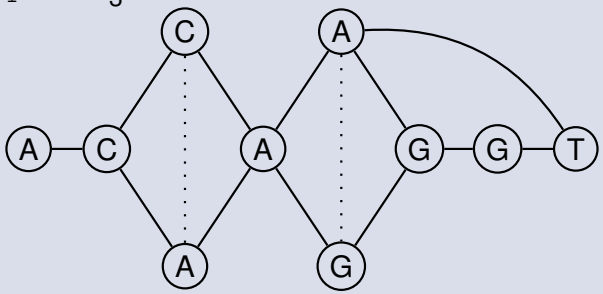
Introduction

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 LoRSCo, a new self-correction method combining both previous strategies:
- LRs are overlapped via a mapping strategy
- Alignments are divided into windows
- Windows consensus are computed using DAGs
- Consensus is polished with the help of local DBGs
- Compared to SOTA: better throughput, comparable quality

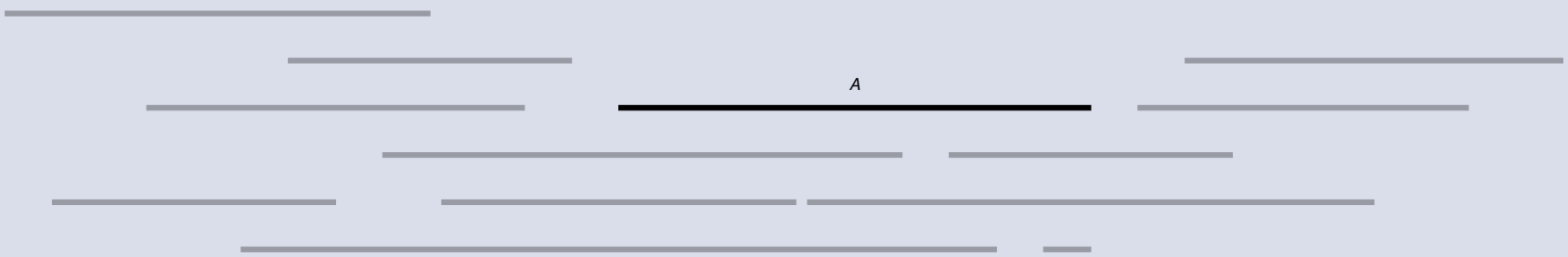
Pre-treatment

Overlap the long reads

Via mapping, with Minimap2 [Li, 2018]

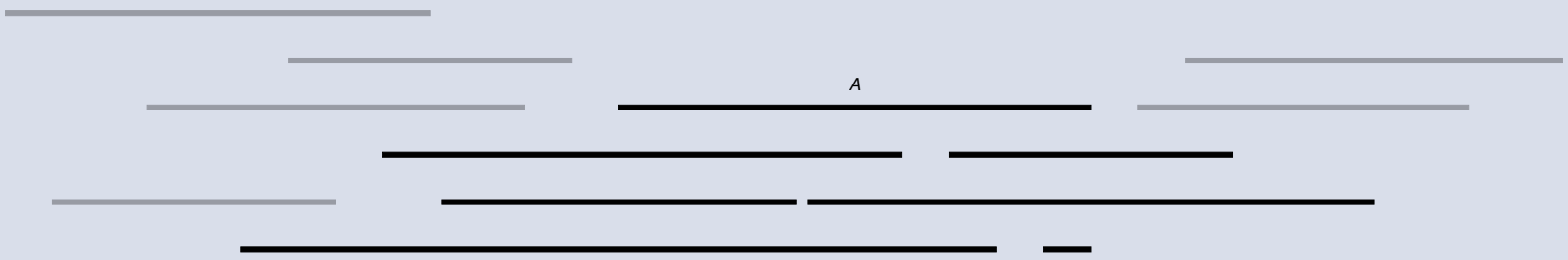
First step: Retrieve alignment pile

Select a long read to correct



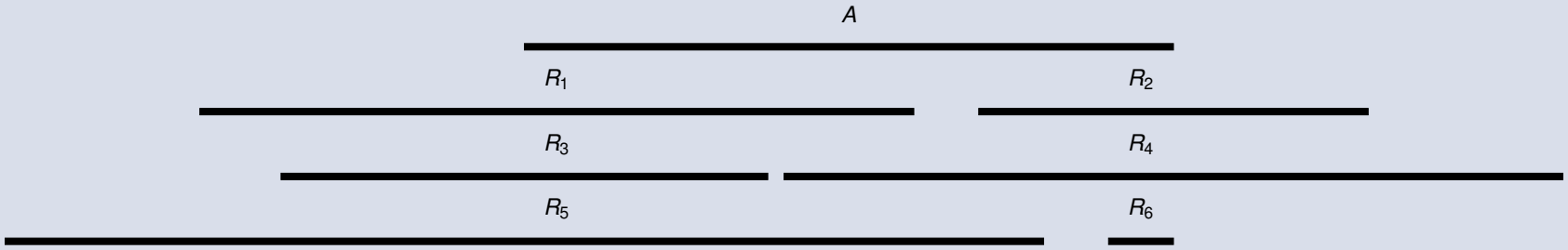
First step: Retrieve alignment pile

Retrieve overlapping long reads



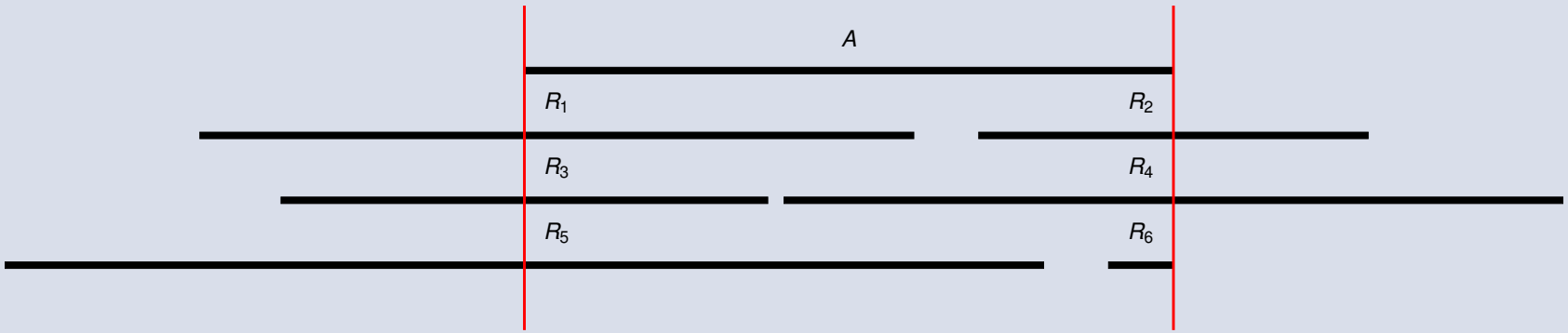
First step: Retrieve alignment pile

Get the alignment pile



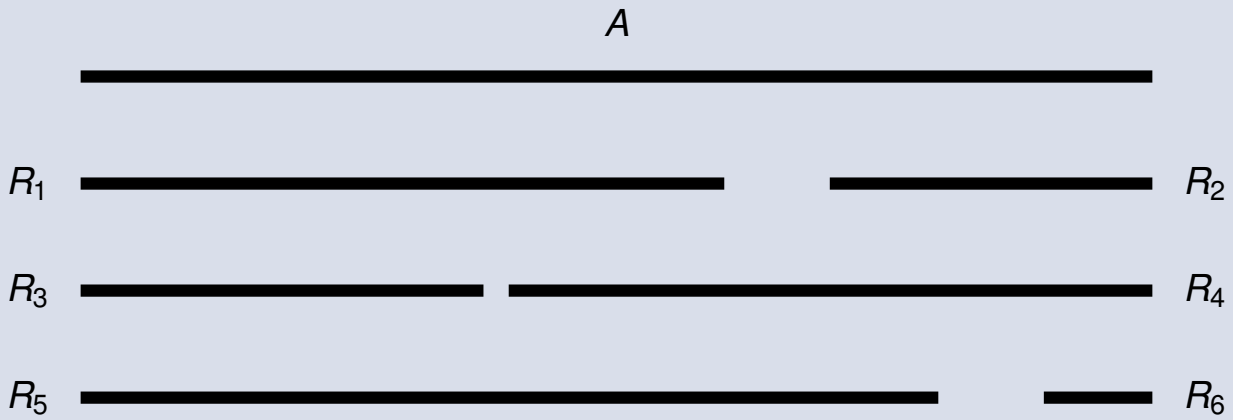
First step: Retrieve alignment pile

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

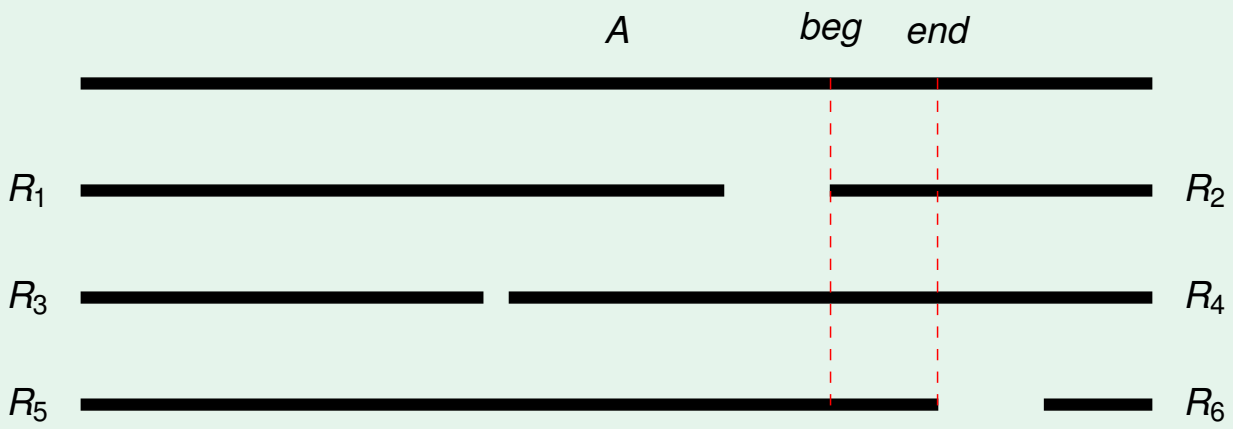
Example

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 $c = 4$:

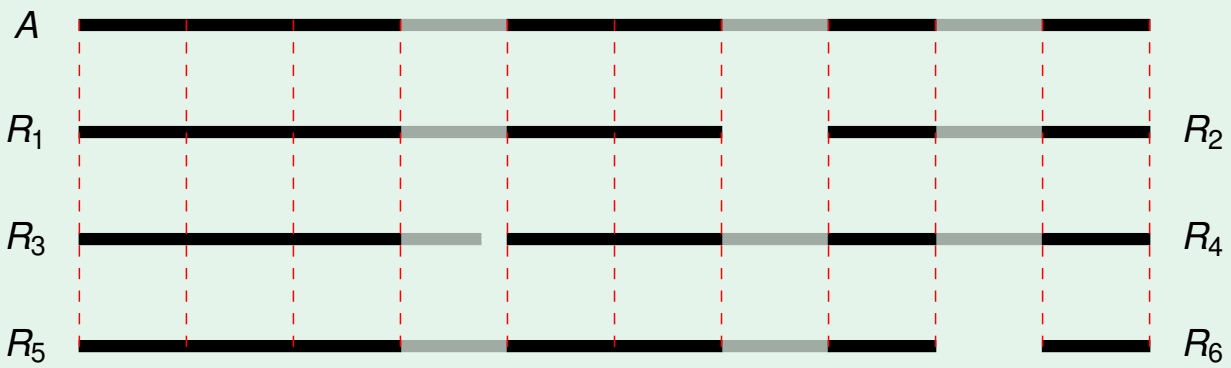
Second step: Divide piles into windows

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

Notations

- We consider the subsequences of reads A, R_1, R_2, ... included in the window
- We call the subsequence of read A the *template* sequence
- We call the subsequences of other reads s_i such as $s_i \in R_i$

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

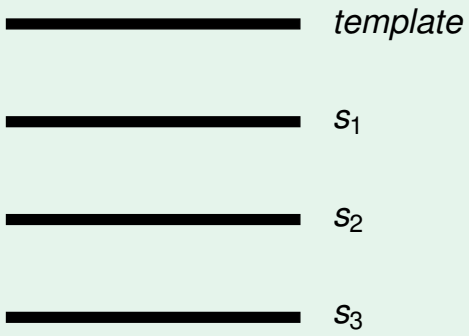
Example (with *solid* = 2 and n = 2)

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

Example (with *solid* = 2 and n = 2)



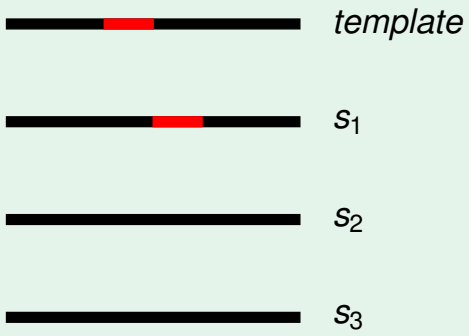
$list = \{template\}$

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

Example (with *solid* = 2 and n = 2)



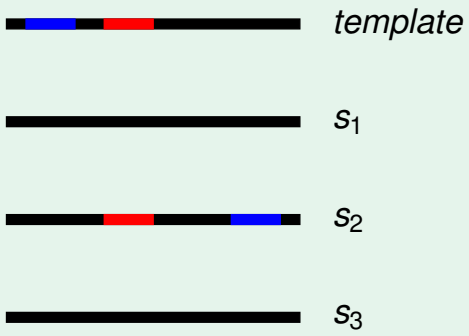
$list = \{template\}$

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

Example (with *solid* = 2 and n = 2)



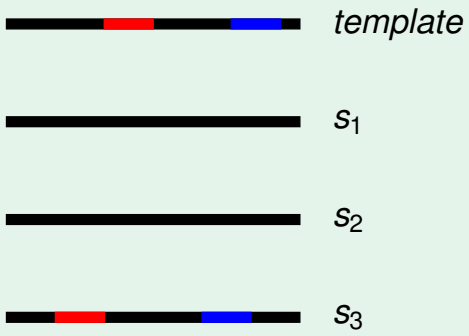
$list = \{template\}$

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

Example (with *solid* = 2 and n = 2)



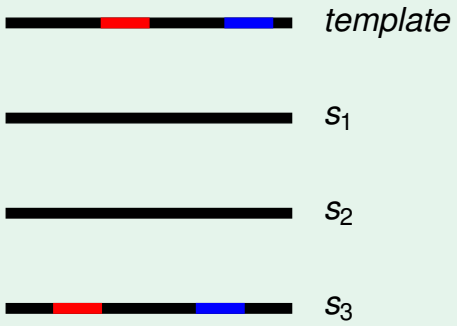
$list = \{template\}$

Third step: Compute consensus of a window

1. Remove bad sequences

- Start with a list containing the *template*
- $\forall i$ if s_i shares n solid, collinear k-mers with the *template*, add s_i to the list

Example (with *solid* = 2 and n = 2)



$list = \{template, s_3\}$

Third step: Compute consensus of a window

2. Compute consensus

- Only consider the sequences of the list
- Compute multiple sequence alignment (MSA) of these sequences
- Compute consensus from the MSA (from the DAG)
- $\Rightarrow$ 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 Smith-Waterman algorithm

Example

Computing alignment of CGATTACG and CGCTTAT

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 Smith-Waterman algorithm

Example

Computing alignment of CGATTACG and CGCTTAT

Third step: Compute consensus of a window

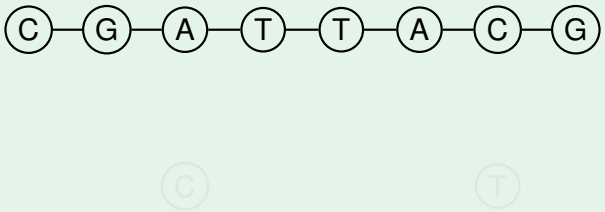
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCTTAT



Third step: Compute consensus of a window

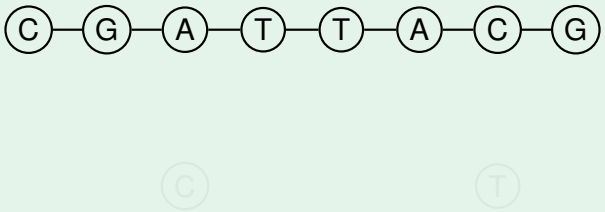
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCTTAT



Third step: Compute consensus of a window

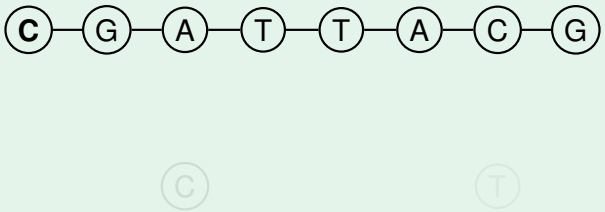
POA

Workflow:

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

Example

Computing alignment of CGATTACG and **C**GCTTAT



Third step: Compute consensus of a window

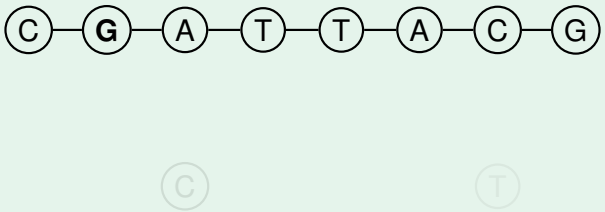
POA

Workflow:

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

Example

Computing alignment of CGATTACG and C**G**CTTAT



Third step: Compute consensus of a window

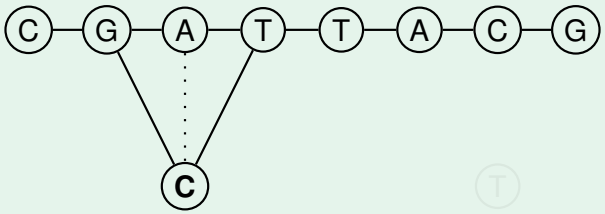
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CG**C**TTAT



Third step: Compute consensus of a window

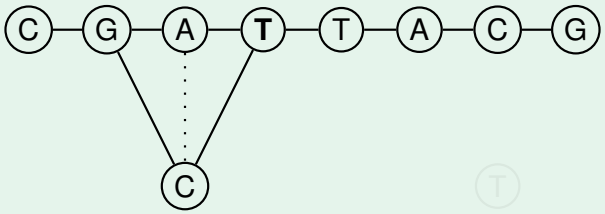
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGC**T**TAT



Third step: Compute consensus of a window

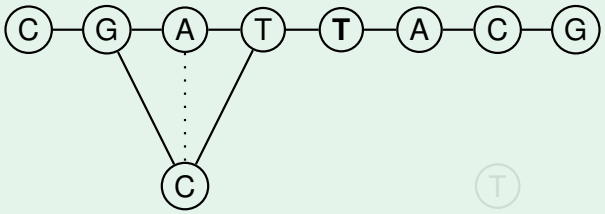
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCT**T**AT



Third step: Compute consensus of a window

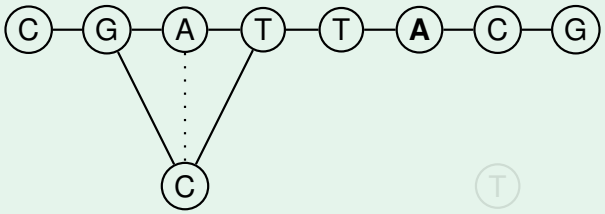
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCTT**A**T



Third step: Compute consensus of a window

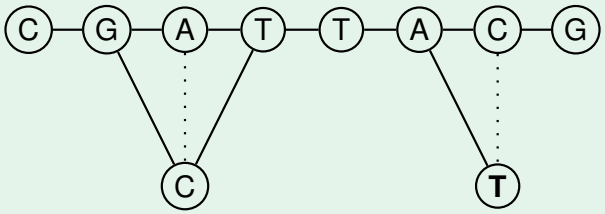
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCTTA**T**



Third step: Compute consensus of a window

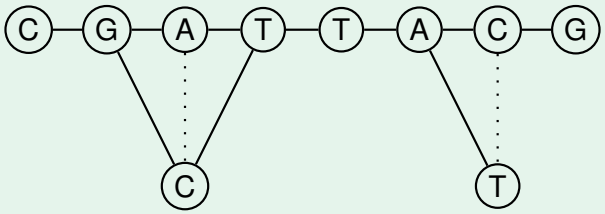
POA

Workflow:

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

Example

Computing alignment of CGATTACG and CGCTTAT



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

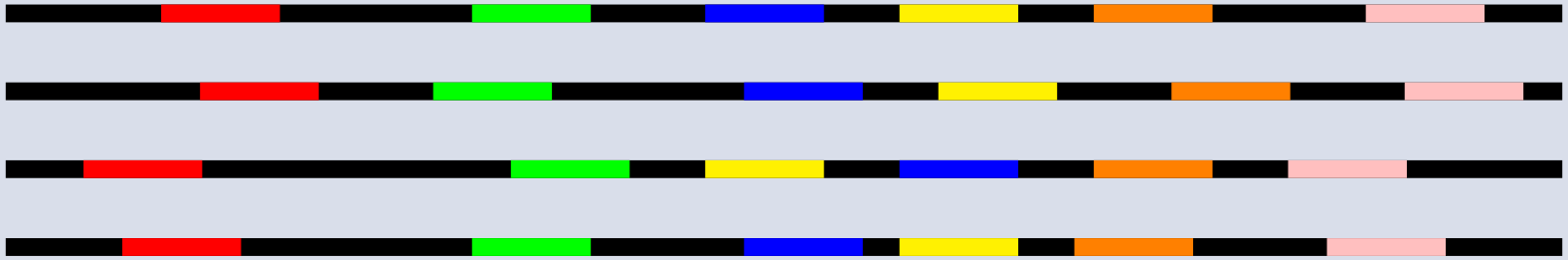
Segmentation strategy

1. Compute shared anchors between the reads

Third step: Compute consensus of a window

Segmentation strategy

1. Compute shared anchors between the reads



Third step: Compute consensus of a window

Segmentation strategy

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

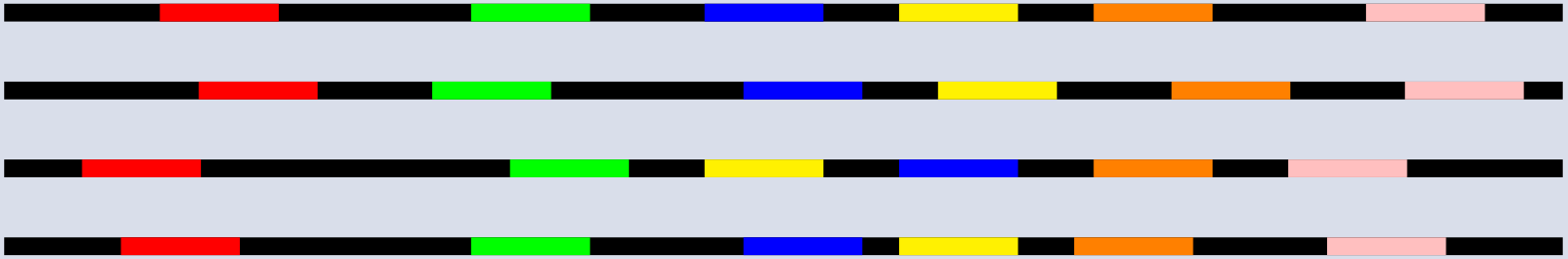
1. A_i is followed by A_{i+1} in at least N reads
2. A_{i+1} is never followed by A_i

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 reads
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 reads
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 reads
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 $\forall A_i\ A_{i+1}$:

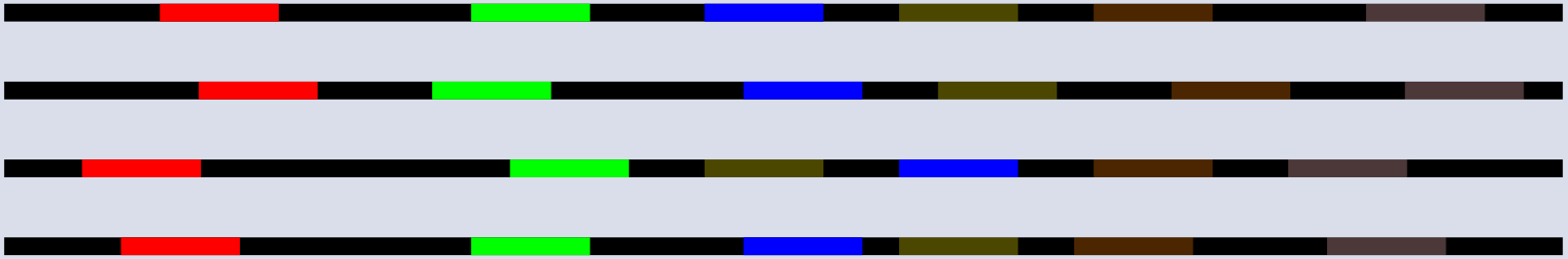
1. A_i is followed by A_{i+1} in at least N reads
2. A_{i+1} is never followed by A_i

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 reads
2. A_{i+1} is never followed by A_i

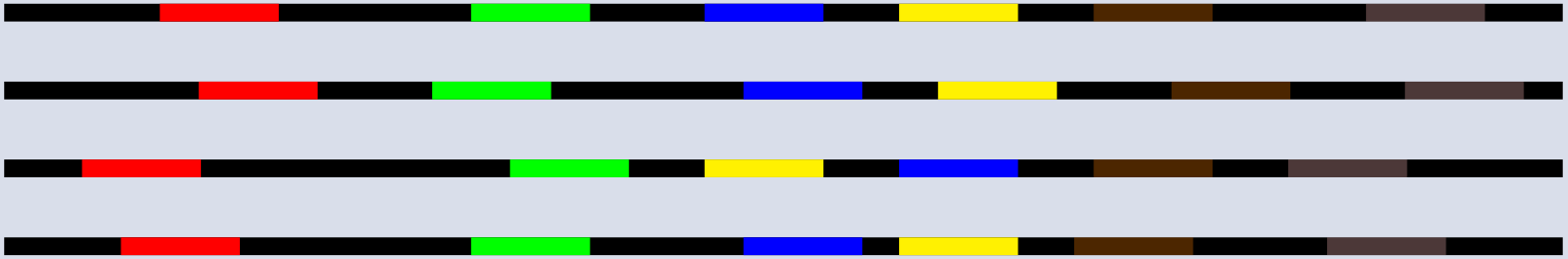


Third step: Compute consensus of a window

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 reads
2. A_{i+1} is never followed by A_i

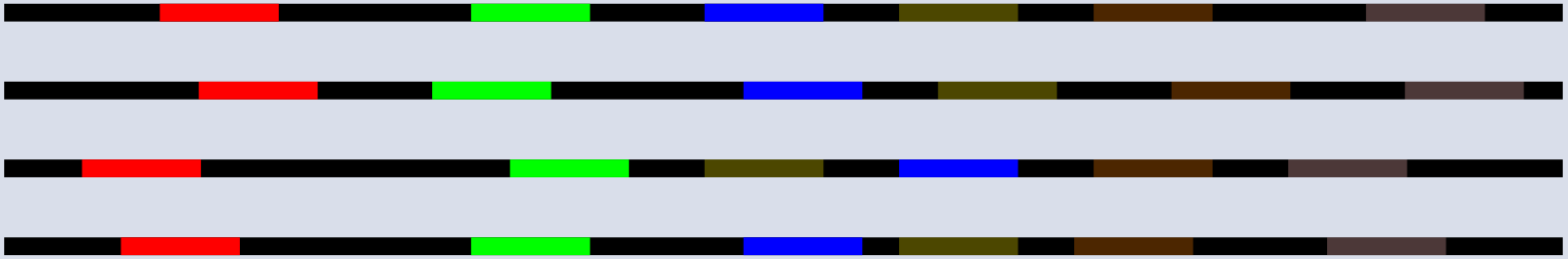


Third step: Compute consensus of a window

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 reads
2. A_{i+1} is never followed by A_i

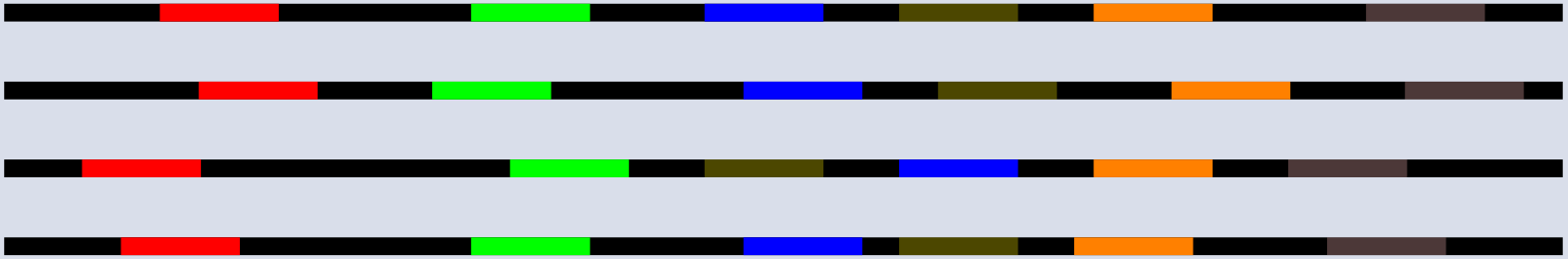


Third step: Compute consensus of a window

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 reads
2. A_{i+1} is never followed by A_i

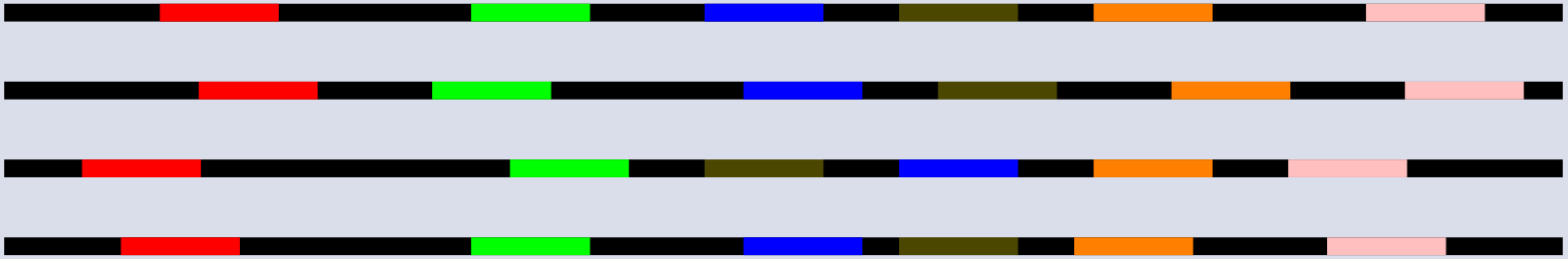


Third step: Compute consensus of a window

Segmentation strategy

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

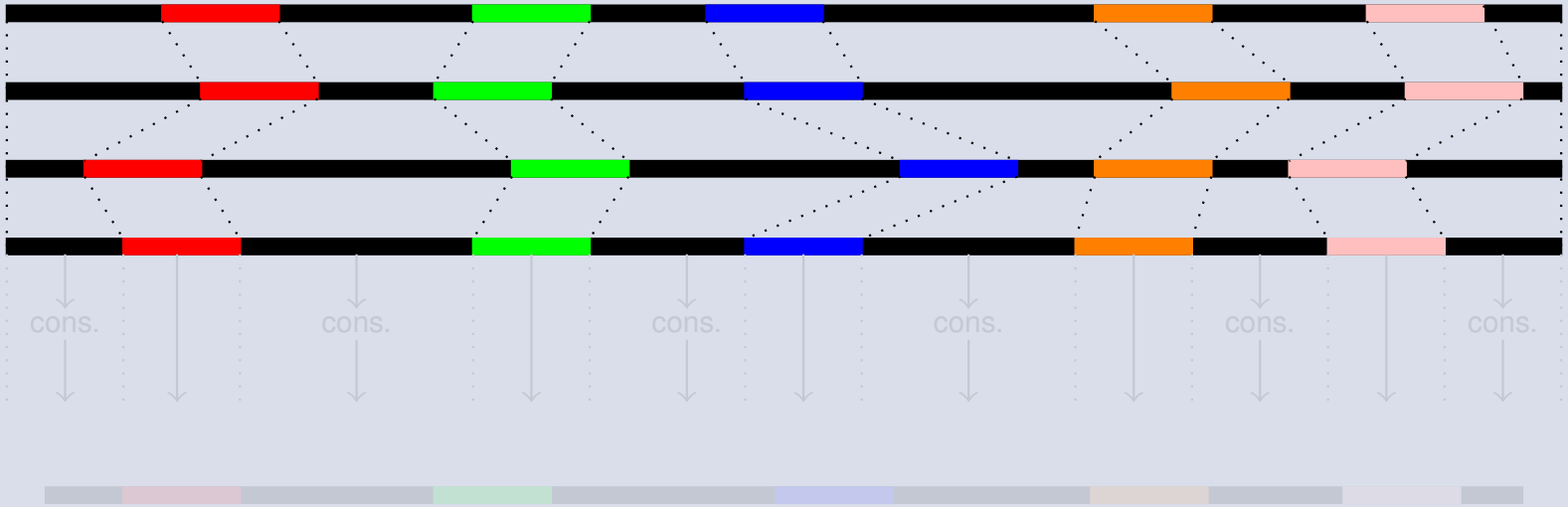
1. A_i is followed by A_{i+1} in at least N reads
2. A_{i+1} is never followed by A_i



Third step: Compute consensus of a window

Segmentation strategy

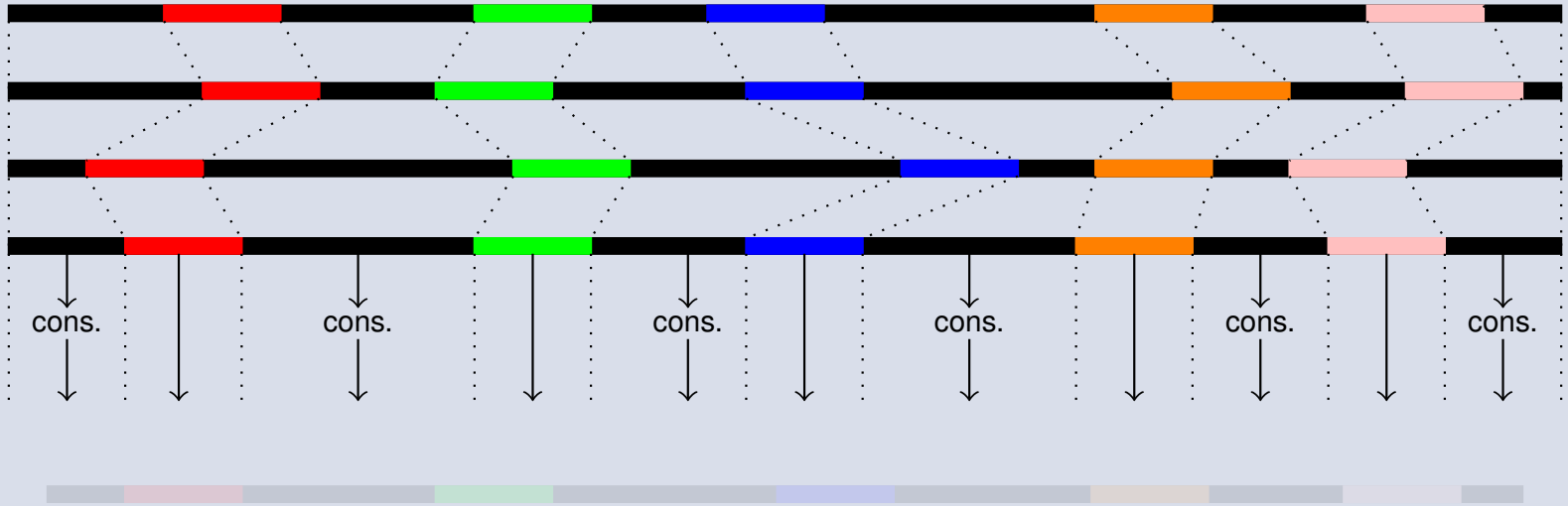
3. Compute MSA / consensus for sequences bordered by anchors



Third step: Compute consensus of a window

Segmentation strategy

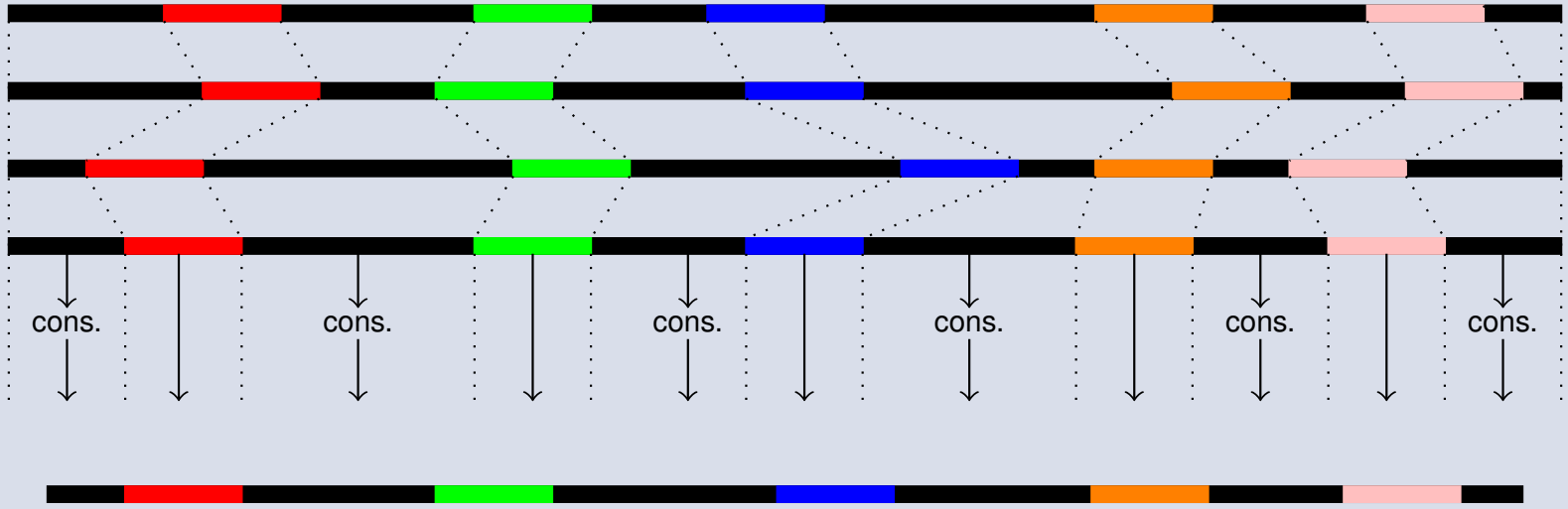
3. Compute MSA / consensus for sequences bordered by anchors



Third step: Compute consensus of a window

Segmentation strategy

3. Compute MSA / consensus for sequences bordered by anchors



Fourth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the consensus result
- Align the *template* to it (with dynamic programming)
- Why not consider the whole consensus? It does not always represent the template...

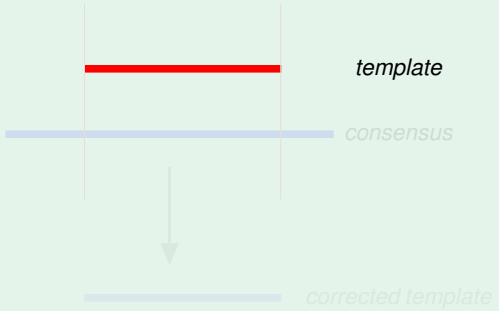
Example

Fourth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the consensus result
- Align the *template* to it (with dynamic programming)
- Why not consider the whole consensus? It does not always represent the template...

Example

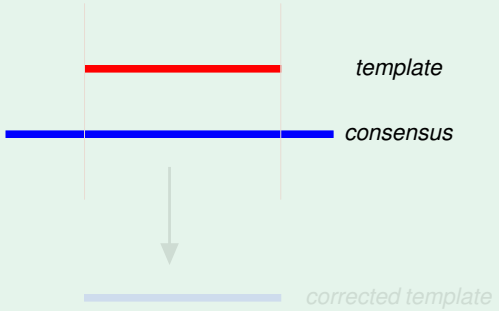


Fourth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the consensus result
- Align the *template* to it (with dynamic programming)
- Why not consider the whole consensus? It does not always represent the template...

Example

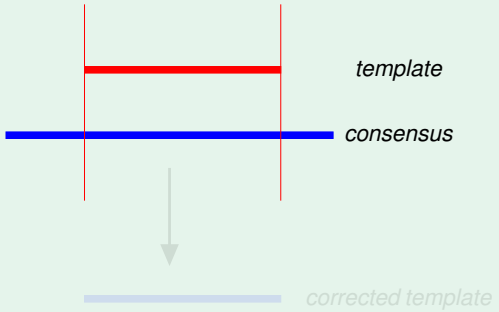


Fourth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the consensus result
- Align the *template* to it (with dynamic programming)
- Why not consider the whole consensus? It does not always represent the template...

Example

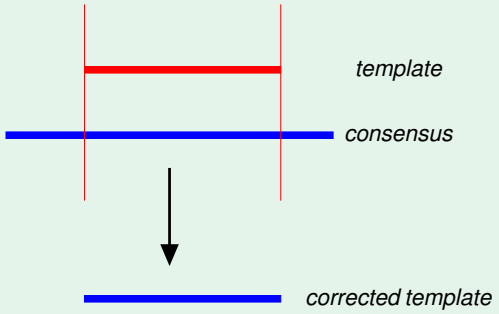


Fourth step: Anchor the consensus to the read

Retrieve the corrected *template*

- Get the consensus result
- Align the *template* to it (with dynamic programming)
- Why not consider the whole consensus? It does not always represent the template...

Example



Fourth step: Anchor the consensus to the read

Align the corrected template to the read

- Replace the aligned part of the template by its correction on the read
- Non-corrected bases in lowercase, corrected bases in uppercase $\Rightarrow$ Polishing
- Repeat with the other windows

Fifth step: Polish the correction

Approach

- Find sketches of lowercase (uncorrected) bases
- Rely on flanking *k*-mers to define a window
- Build a DBG from the window's sequences
- Traverse the graph to find a path between the anchor *k*-mers

```
...GATCGGGTcatTGCCCGTGTTTATGCGTGTG...
```

Fifth step: Polish the correction

Approach

- Find sketches of lowercase (uncorrected) bases
- Rely on flanking *k*-mers to define a window
- Build a DBG from the window's sequences
- Traverse the graph to find a path between the anchor *k*-mers

```
...GATCGGGTcatTGCCCGTGTTTATGCGTGTG...
```

Experiments

Datasets

- *E. coli*, 50x PacBio simulated LRs, 12% error rate
- *S. cerevisiae*, 50x PacBio simulated LRs, 12% error rate

Compared tools

- Canu
- Daccord
- LoRMA
- MECAT

Experiments

Datasets

- *E. coli*, 50x PacBio simulated LRs, 12% error rate
- *S. cerevisiae*, 50x PacBio simulated LRs, 12% error rate

Compared tools

- Canu
- Daccord
- LoRMA
- MECAT

Experiments

Results (*E. coli*)

Corrector	Throughput (Mbp)	Error rate (%)	Runtime	Memory peak (MB)
Original	232	12.2674	N/A	N/A
Canu	173	0.5841	19 min 20	3,623
daccord	**218**	**0.0166**	38 min	13,559
LoRMA	126	9.4315	37 min	31,902
MECAT	193	0.1118	**4 min**	**2,130**
LRSC	211	0.1784	1 h	3,927

Experiments

Results (*S. cerevisiae*)

Corrector	Throughput (Mbp)	Error rate (%)	Runtime	Memory peak (MB)
Original	618	12.2835	N/A	N/A
Canu	477	0.6294	55 min	**3,702**
daccord	**579**	**0.0451**	1 h 51 min	31,774
LoRMA	339	9.6010	2 h 41 min	31,480
MECAT	510	0.1493	**11 min**	4,275
LRSC	561	0.3412	3 h 56 min	8,487

Conclusion

- Combines different strategies from the SOTA
- Computes actual MSA
- Introduces a segmentation strategy allowing fast computation of MSA
- Compares well to the SOTA
- Runtime remains an issue
- Available at: `https://github.com/morispi/LoRSCo`

Future works

- Focus on the runtime:
  - Adapt the parameters
  - Optimize the polishing step
- Adapt windows length and coverage threshold in real time
- Validate the method on real data

Thanks for your attention

References I

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.

References II

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. (2017).
Accurate selfcorrection of errors in long reads using de Bruijn graphs.
Bioinformatics, 33:799–806.

References III

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.