Pangenome graph augmentation from unassembled long reads

Luca Denti*, *Paola Bonizzoni, Brona Brejova, Rayan Chikhi, Thomas Krannich, Tomas Vinar, and Fereydoun Hormozdiari*

**Comenius University, Bratislava*

ITAT - WBCB, 2025

ROBERT KOCH INSTITUT

UNIVERZITA KOMENSKÉHO V BRATISLAVE

UCDAVIS UNIVERSITY OF CALIFORNIA

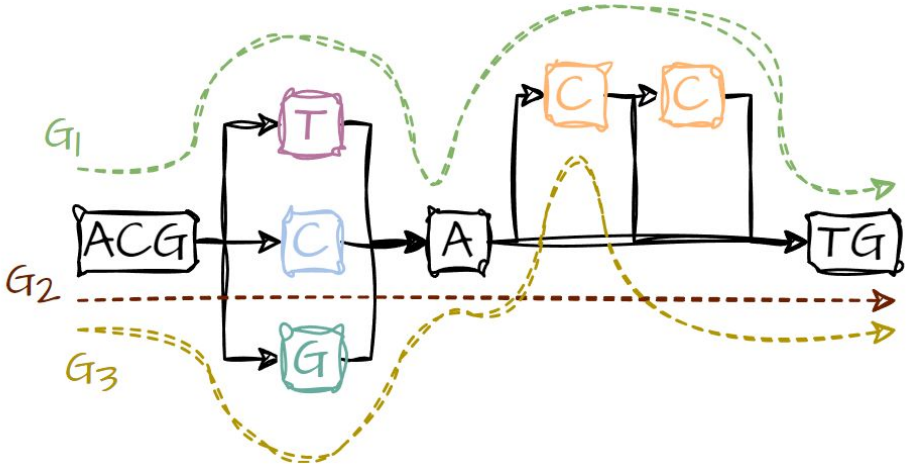
INSTITUT PASTEUR

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

What is a pangenome?

A pangenome is a collection of genomes

G_1 = ACGTACCTG
G_2 = ACGCATG
G_3 = ACGGACTG



*"most **[graph] paths** are unlikely recombinations of true haplotypes"**

genome == haplotype == assembly == path

** Sirén et al. **Personalized pangenome references.** Nature Methods (2024)*

Why pangenomes?

*"No single genome can **represent the diversity** in the human population. Using a single reference genome creates **reference biases**, adversely affecting variant discovery, gene–disease association studies and the accuracy of genetic analyses."*

https://humanpangenome.org/

How to build a pangenome?

MSA-based:

- `make_prg/Pandora` *[Colquhoun et al. 2021]*
- `panpa` *[Dabbaghie et al. 2023]*
- `pangeblocks` *[Avila et al. 2024]*

All-vs-all:

- `pggb` *[Garrison et al. 2024]*

Progressive:

- `minigraph` *[Li et al. 2020]*
- `minigraph-cactus` *[Hickey et al. 2024]*

How to build a pangenome?

MSA-based:

- `make_prg/Pandora` *[Colquhoun et al. 2021]*
- `panpa` *[Dabbaghie et al. 2023]*
- `pangeblocks` *[Avila et al. 2024]*

All-vs-all:

- `pggb` *[Garrison et al. 2024]*

Progressive:

- `minigraph*` *[Li et al. 2020]*
- `minigraph-cactus` *[Hickey et al. 2024]*

High-quality assemblies:

✗ *costly (multiple seq.tech.)*

✗ *computationally expensive*

**Minigraph can work with reads, but construction pipeline has been designed for assemblies.*

Progressive construction

A graph can be progressively constructed by iteratively aligning and inserting one genome at the time

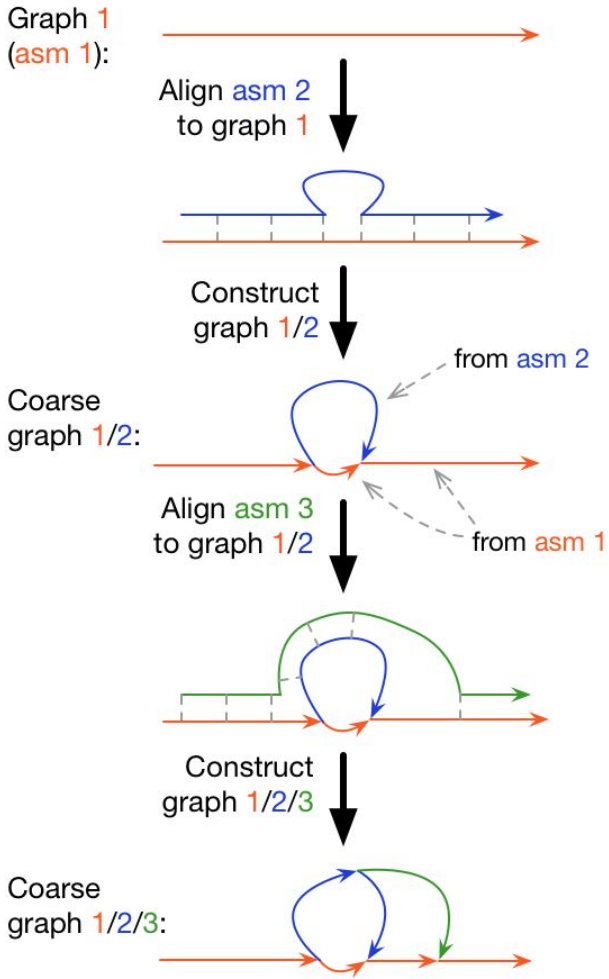
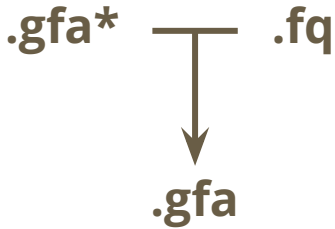


Image from github.com/lh3/minigraph

Progressive construction from long reads

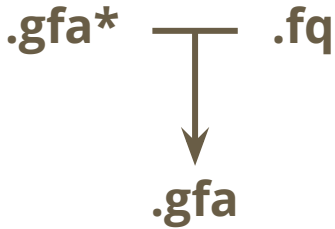
Provide an **assembly-free** and **alignment-free** approach for the progressive construction of a pangenome



***Assumption:** DAG*

Progressive construction from long reads

Provide an **assembly-free** and **alignment-free** approach
for the progressive construction of a pangenome



Several questions:

how can we…

pinpoint differences with known genomes?

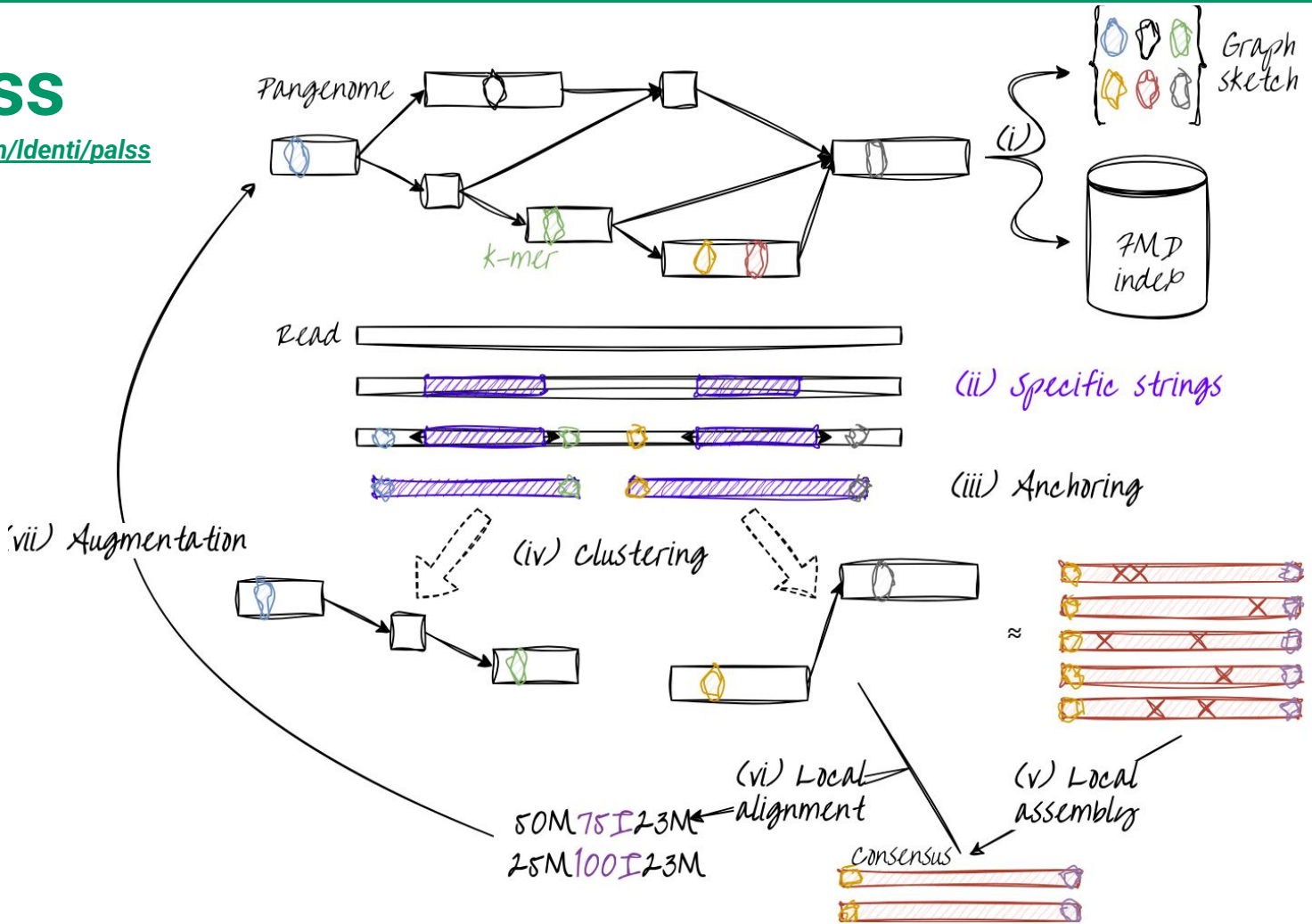
locate such differences in the graph?

summarize information coming from different reads?

***Assumption:** DAG*

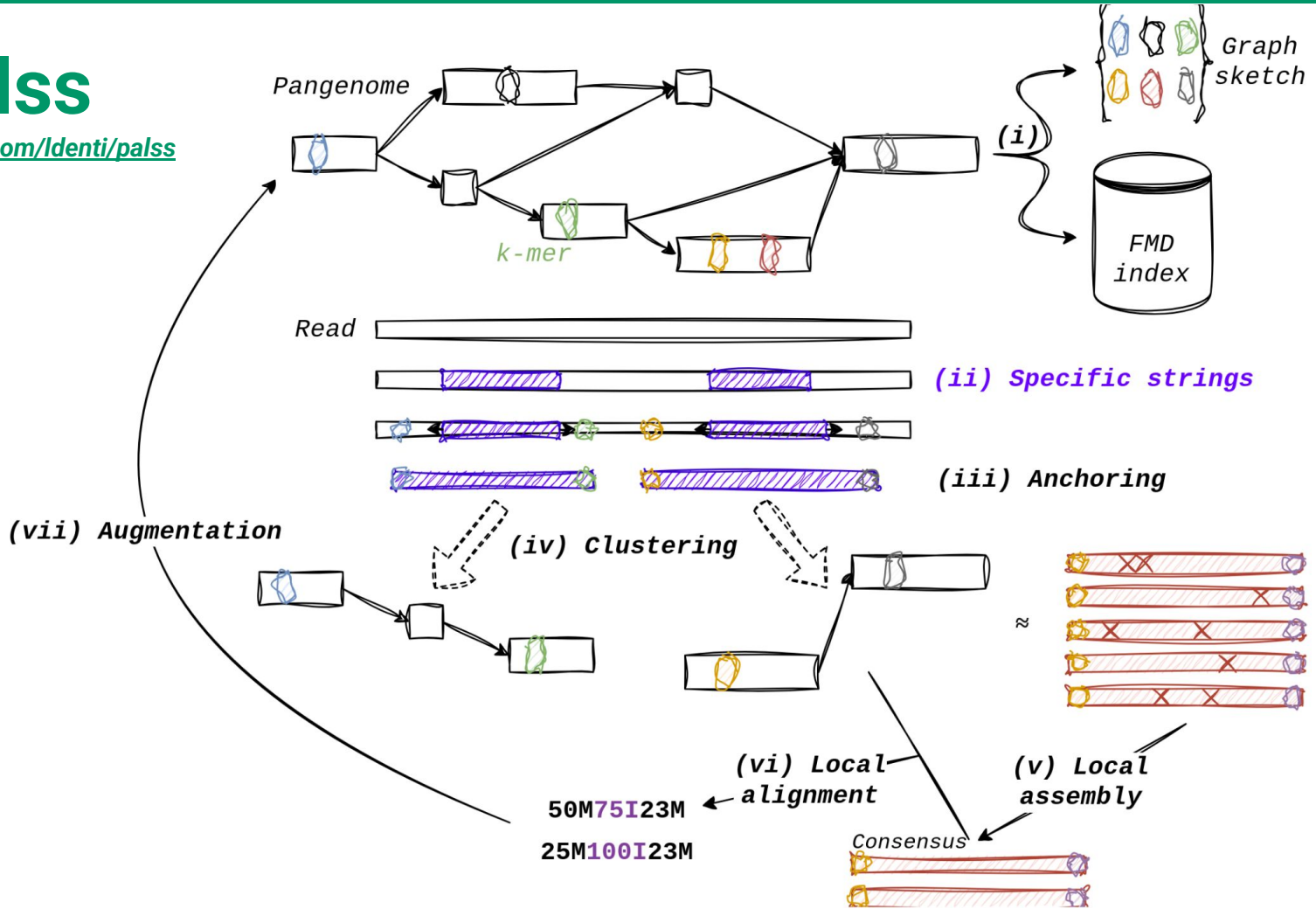
palss

github.com/Identi/palss



palss

github.com/Identi/palss



Experimental evaluation

Experiments on real pangenomes and simulated read samples to evaluate:

1. Reliability of sketching scheme
    - *can we actually use our solid anchors?*

2. Accuracy of graph augmentation
    - *can we trust our results?*

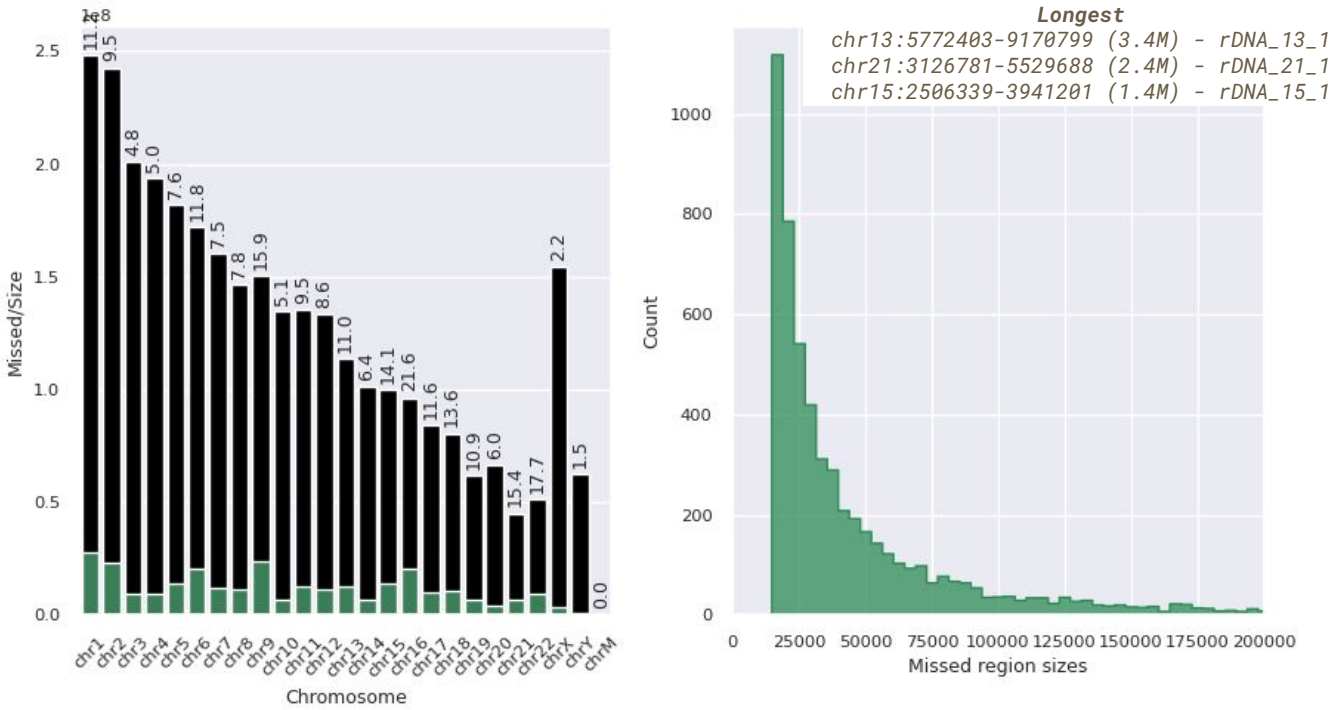
Exp1 - Reliability of sketching scheme

1. Full HPRC graph v1.1 on T2T (minigraph-cactus)
2. Sketch graph *(with k=27)*
3. Check distance between consecutive solid anchors on T2T genome

(we miss all regions longer than avg read length, e.g., 15k for HiFi)

We miss 9.3% of the genome *(HPRC T2T)*

1.79/2.59B (69%) total solid anchors (94.4% consecutive pairs are *≤100bp apart*)



5588 missed regions *intersecting known challenging regions*

(repeats, segmental duplications, and centromeric)

Exp1 - Reliability of sketching scheme

1. Full HPRC graph v1.1 on T2T (minigraph-cactus)
2. Sketch graph *(with k=27)*
3. Check distance between consecutive solid anchors on T2T genome

   (we miss all regions longer than avg read length)

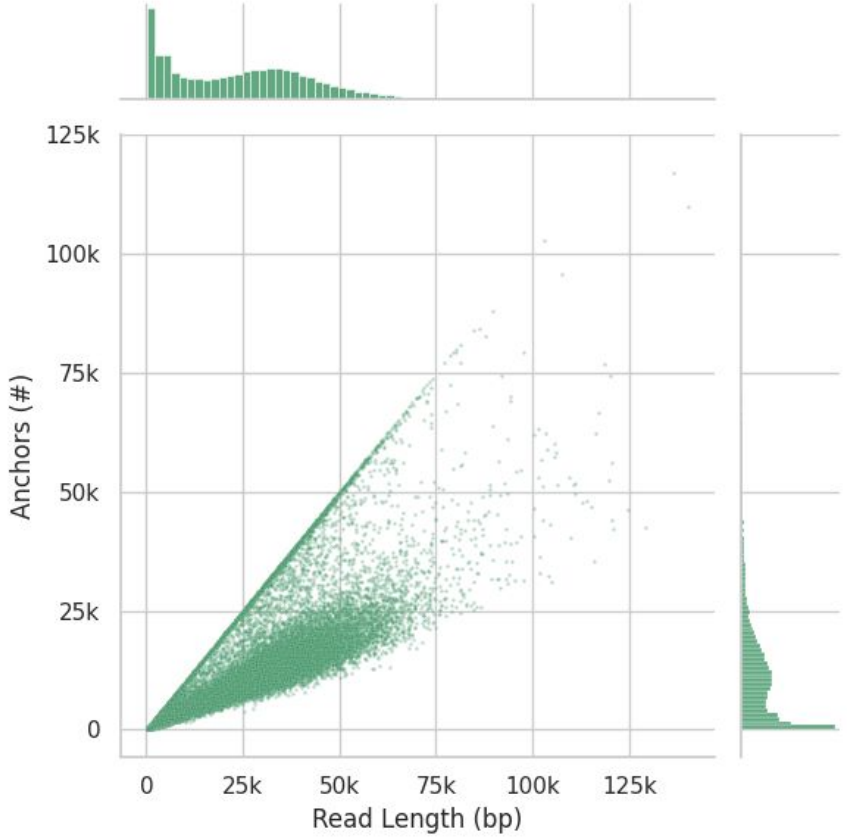
4. Check how many anchors we can find in real reads

   (we miss all reads with <2 anchors)

We can anchor real reads

R10.4 simplex ONT 10× from chr20*
(26 198 reads)

- All reads have at least 2 anchors
  - 100% with at least 29
  - 99% with at least 124
  - 98% with at least 167



*Kolesnikov et al. Nature Communications (2024)

Exp2 - Accuracy of graph augmentation

1. Build graphs with increasing number **n** of individuals *(HPRC chr20 GFA on T2T)*
2. Simulate diploid HiFi reads from individual **x** not in graph *(leave-1-out)*
3. Augment the graph with corrected reads *(hifiasm ec)*
4. Align reads to **graph** with graphaligner

Exp2 - Accuracy of graph augmentation

1. Build graphs with increasing number **n** of individuals
2. Simulate diploid HiFi reads from individual **x** not in graph
3. Augment the graph with corrected reads
4. Align reads to **graph** with graphaligner

- *Original graph with n+x* *FULL*
- *Original graph with n* *1OUT*
- *Augmented n graph* *1OUT-AUG*

Exp2 - Accuracy of graph augmentation

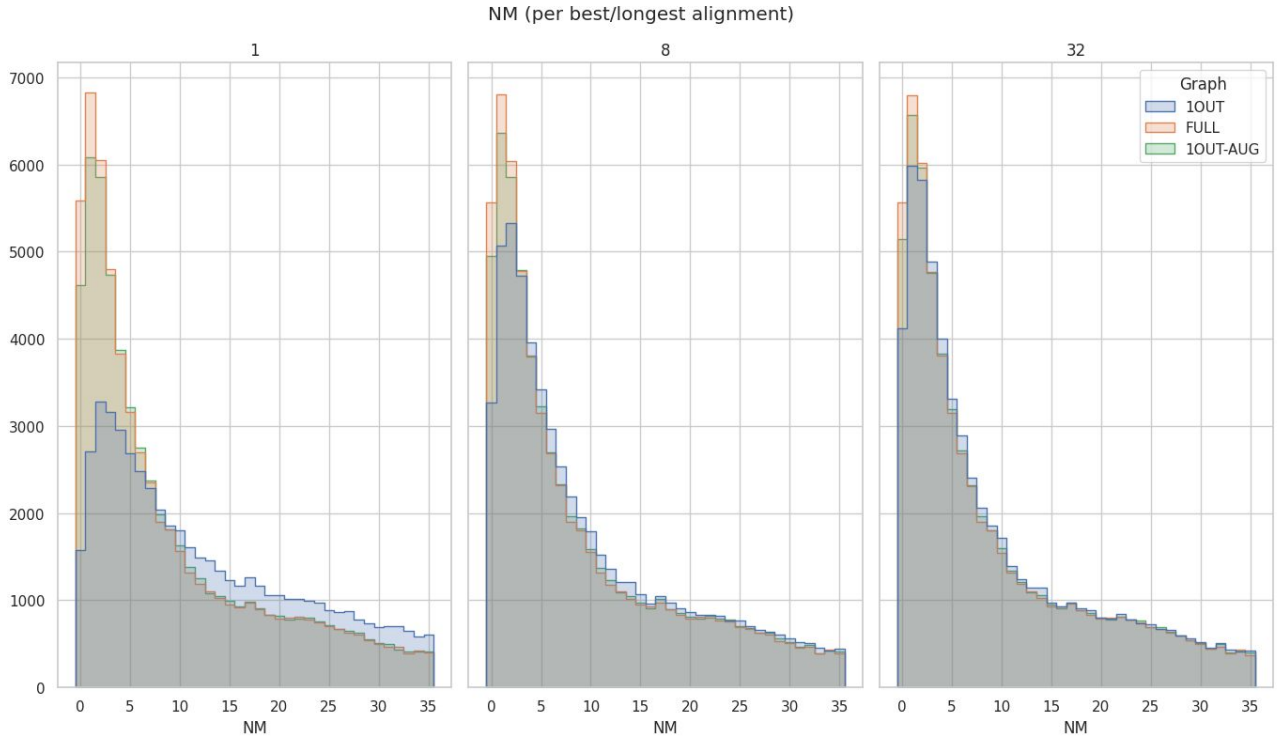
1. Build graphs with increasing number **n** of individuals
2. Simulate diploid HiFi reads from individual **x** not in graph
3. Augment the graph with corrected reads
4. Align reads to **graph** with graphaligner

- *Original graph with n+x* *FULL*
- *Original graph with n* *1OUT*
- *Augmented n graph* *1OUT-AUG*

Metrics:

- “recall” : *alignments accuracy*
- “precision” : *percentage of novel vertices used by the alignments*

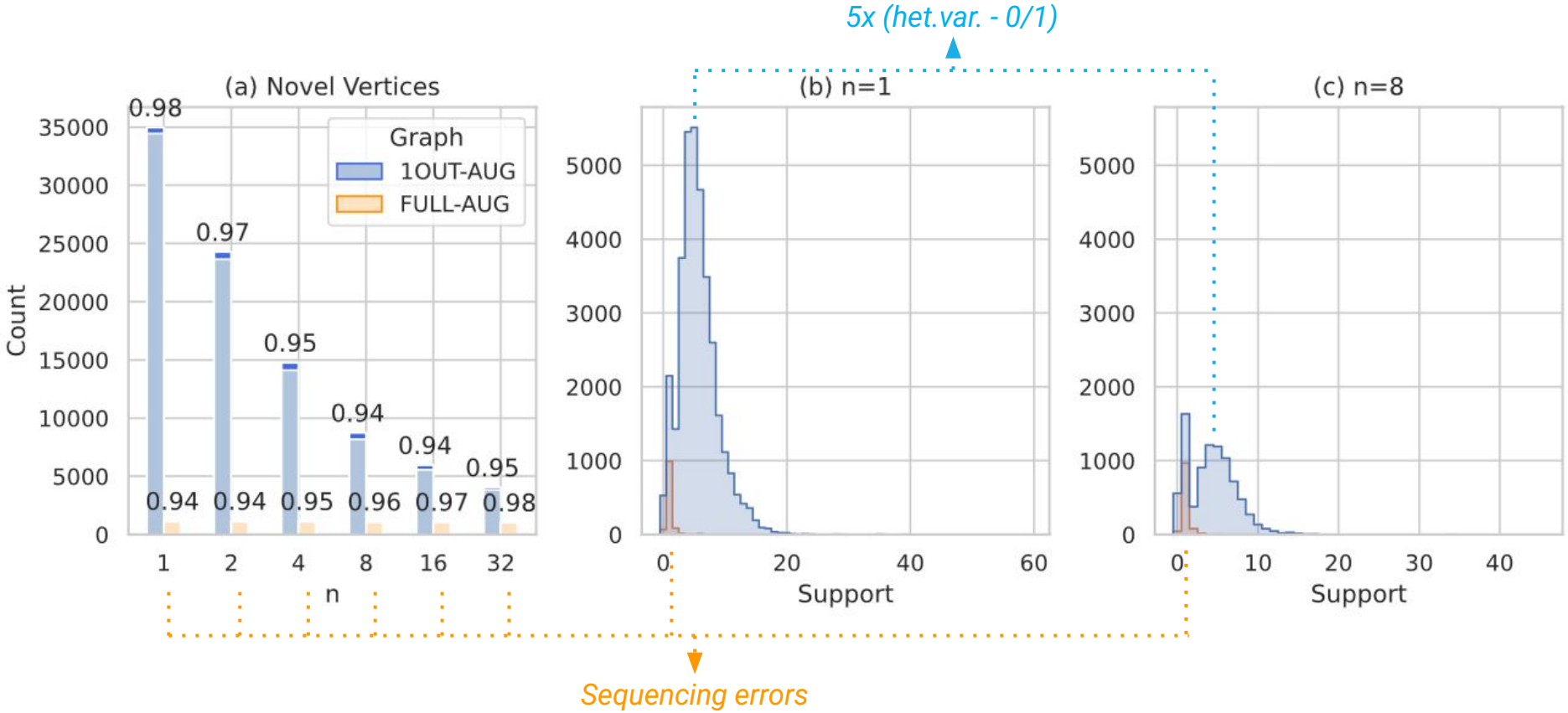
Exp2 - Alignment accuracy *(chr20, 10x)*



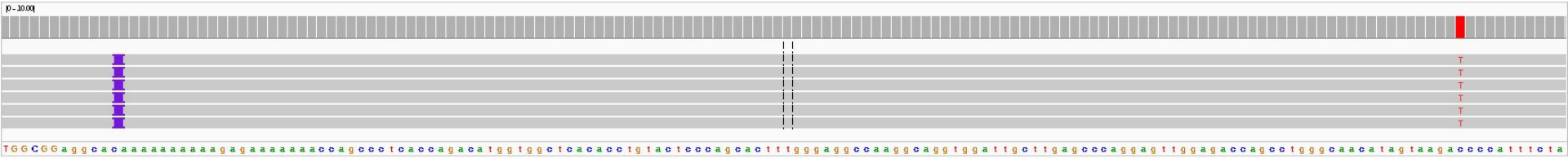
palls augmentation improves read alignments and produces graphs more similar to FULL

Exp2 - Novel vertices

(chr20, 10x)

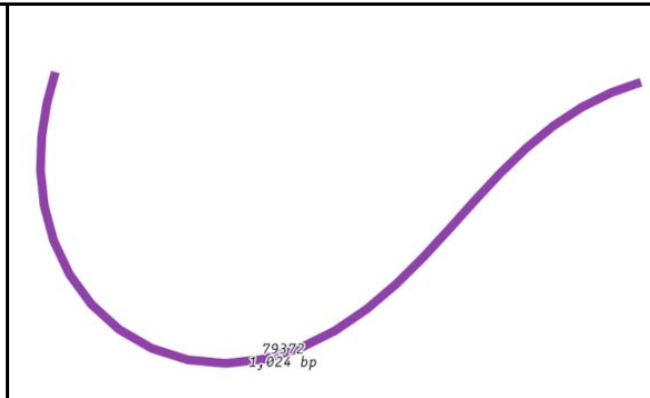


Comparison with Minigraph-Cactus *(chr20)*

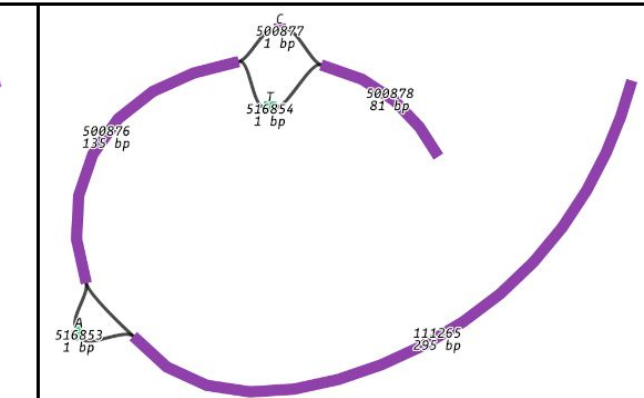


(a) Read Information

(b) 10UT graph



(c) MGC graph



(d) 10UT–AUG graph

- *de novo assembly misses 15 challenging regions (telomeric and centromeric) with 65 specific variants*
- *our approach can also augment regions that are challenging for assemblers*

Efficiency

FMD-indexing (HPRC graph)	28.7 hours (32 threads), 92GB
Sketching (HPRC graph)	1 hour (32 threads), 62GB

One time expenses

Augmentation of (n=)32 chr20:	4.5 hours (1 thread), 32GB
MGC of (n=)32 chr20:	7 hours (16 thread) without assembly step

Conclusions

10.1101/2025.02.07.637057

github.com/Identi/palss

Pangenome graphs can be augmented directly from long reads

*without requiring **expensive high-quality** genome assemblies*

Work-in-progress:

- improve accuracy (new anchor definition) and scalability
- remove DAG constraint *(use HPRC graph)*
- evaluation on real sample
- local augmentation ➡ full haplotype paths *(realignment)*
- variant calling

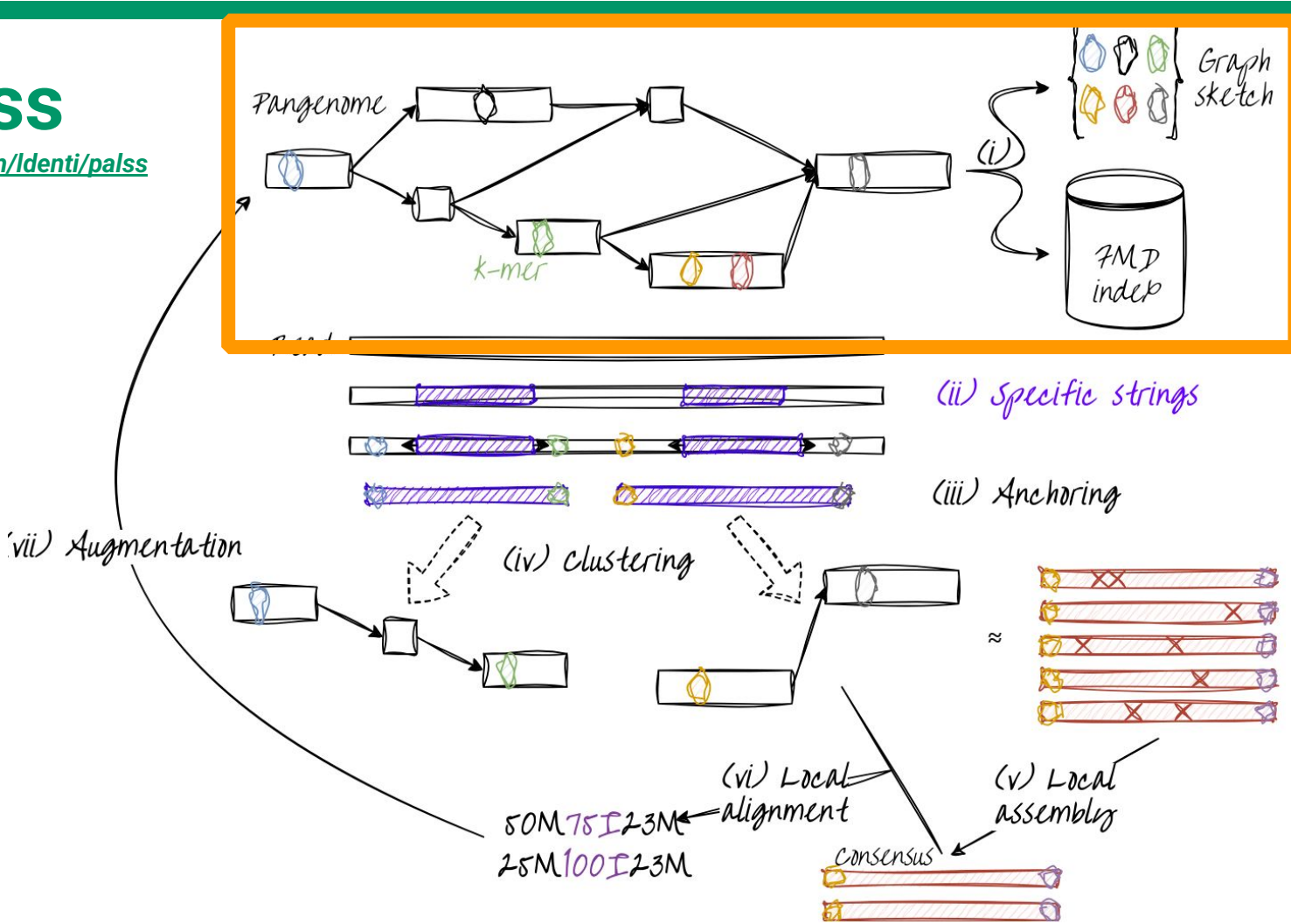
MINISTERSTVO ŠKOLSTVA, VÝSKUMU, VÝVOJA A MLÁDEŽE SLOVENSKEJ REPUBLIKY
Grants 1/0538/22, 1/0140/25

U.S. National Science Foundation
Award DBI-2042518

Funded by the European Union
ASVA-CGR No.101180581
PANGAIA No. 872539

palss

github.com/Identi/palss



Path indexing and graph sketching

We rely on **both representations** of a pangenome:

- **string-based**
  - *paths (i.e., haplotypes) are indexed using an FMD-index**
- **graph-based**
  - *graph is sketched using **solid anchors*** *(anchor ↦ <vertex,offset> – no hashmap though)*

*H. Li. ***BWT construction and search at the terabase scale.*** *Bioinformatics (2024)*

Path indexing and graph sketching

We rely on **both representations** of a pangenome:

- **string-based**
  - *paths (i.e., haplotypes) are indexed using an FMD-index*
- **graph-based**
  - *graph is sketched using **solid anchors*** *(anchor ↦ <vertex,offset> – no hashmap though)*

↓

canonical k-mers occurring in only one vertex of the graph and less than #paths times in the FMD-index

(vertex space ≠ path space)

FM-Index

- **Self-index** for strings
  - data + index in a compressed structure
  - small size with full-text searching
  - based on BWT permutation
  - counts array + rank structure(s)

	SA	Rotations	BWT	$	a	b	n
0	7	$banana	a	0	1	0	0
1	6	a$banan	n	0	1	0	1
2	4	ana$ban	n	0	1	0	2
3	1	anana$b	b	0	1	0	2
4	0	banana$	$	1	1	1	2
5	5	na$bana	a	1	2	1	2
6	2	nana$ba	a	1	3	1	2
Counts				0	1	4	5

- Operations on pattern P in linear time *O(|P|)*:
  - **count(P)**: thanks to LF-mapping (aka **backward extension** in *O(1)*)
  - locate(P): with the addition of Suffix Array (very expensive, *sampling*)

FMD-Index

FM-Index of a *bidirectional collection* of DNA sequences:

$$R_1 \cdot \mathbf{RC}(R_1) \cdot R_2 \cdot \mathbf{RC}(R_2) \cdot \ldots \cdot R_n \cdot \mathbf{RC}(R_n)$$

RC(R) is reverse-and-complement R

Single index for both forward and reverse strands that allows in O(1):

➔ backward extensions

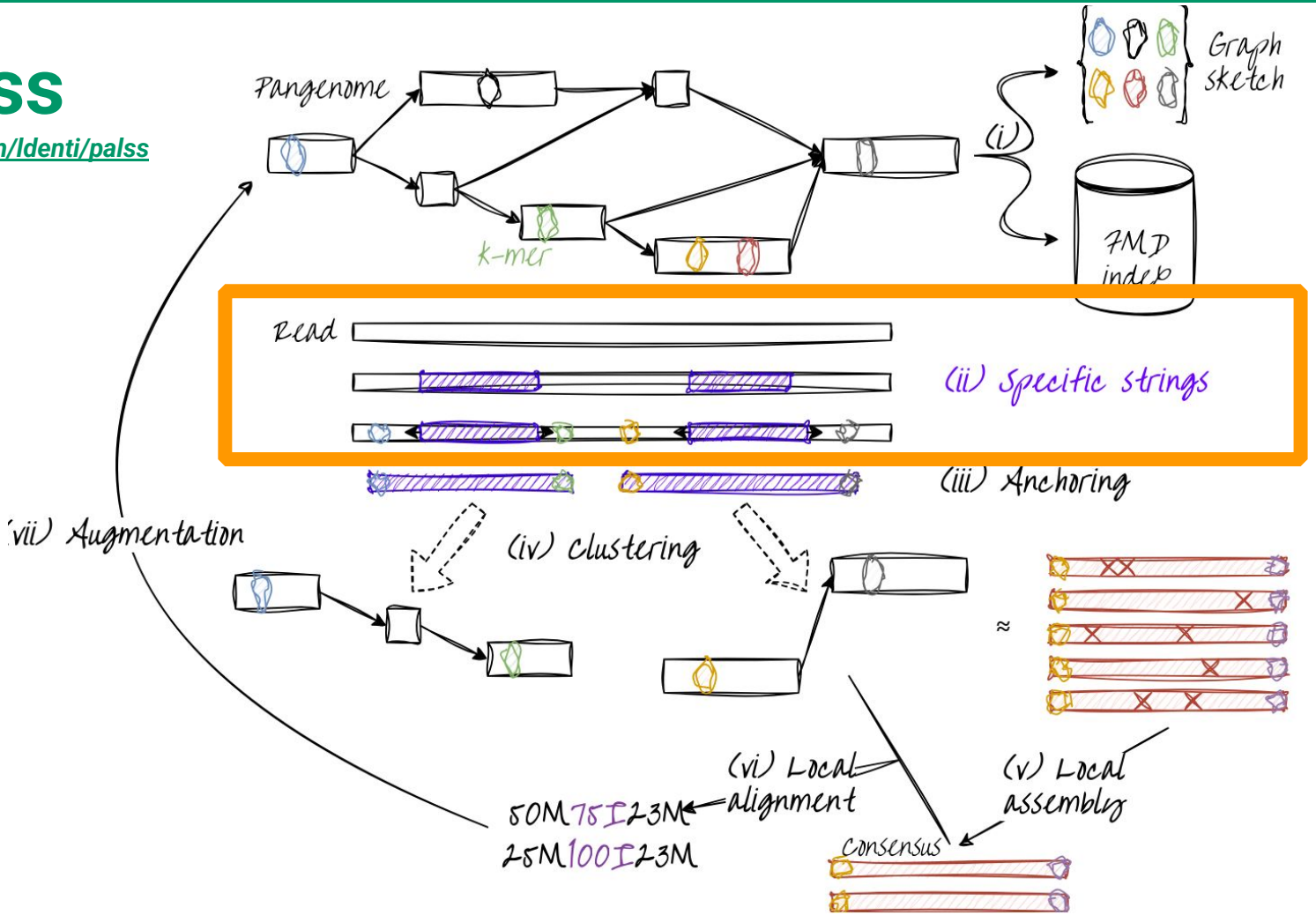
➔ **forward extensions**

*by backward extending pattern P with **A**, we also forward extend P with **T***

Heng Li. ***Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly.*** *Bioinformatics (2012)*
Heng Li. ***Fast construction of FM-index for long sequence reads.*** *Bioinformatics (2014)*
Heng Li. ***BWT construction and search at the terabase scale.*** *Bioinformatics (2024)*

palss

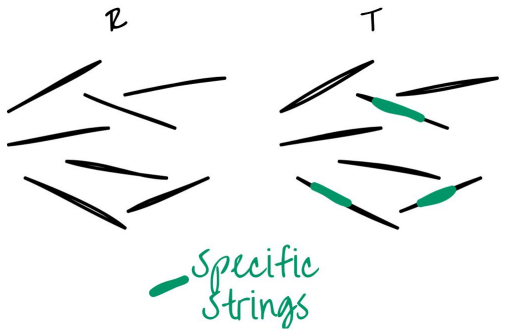
github.com/Identi/palss



Pinpoint differences with specific strings*

Given two sets of strings *(R,T)*, a **specific string** is any substring:

I. occurring in T and not occurring in R
II. s.t. no other specific string is contained in it *(i.e., it is the shortest)*



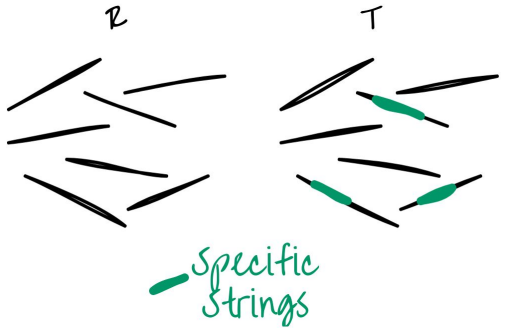
Specific strings pinpoint any difference between the two sets of strings

*P. Khorsand, L. Denti, et al. Bioinformatics Advances (2021)

Pinpoint differences with specific strings*

Given two sets of strings *(R,T)*, a **specific string** is any substring:

I. occurring in T and not occurring in R
II. s.t. no other specific string is contained in it *(i.e., it is the shortest)*



Specific strings pinpoint any difference between the two sets of strings

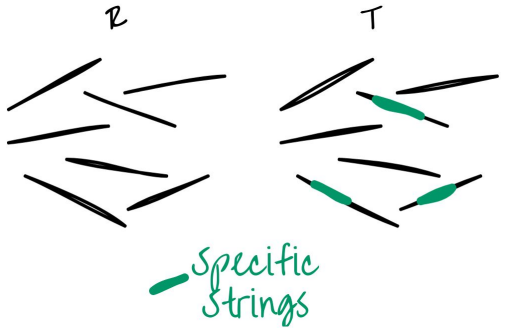
*if we index a reference genome, they pinpoint (mostly) all variations of a sample***

*P. Khorsand, L. Denti, et al. Bioinformatics Advances (2021)
**L. Denti, P. Khorsand, et al. Nature Methods (2023)

Pinpoint differences with specific strings*

Given two sets of strings *(R,T)*, a **specific string** is any substring:

I. occurring in T and not occurring in R
II. s.t. no other specific string is contained in it *(i.e., it is the shortest)*



Specific strings pinpoint any difference between the two sets of strings

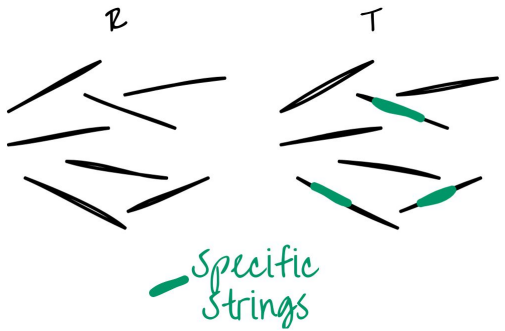
*if we index a reference **pan**genome, they pinpoint (mostly) all **novel** variations of a sample*

*P. Khorsand, L. Denti, et al. Bioinformatics Advances (2021)

Pinpoint differences with specific strings*

Given two sets of strings *(R,T)*, a **specific string** is any substring:

I. occurring in T and not occurring in R
II. s.t. no other specific string is contained in it *(i.e., it is the shortest)*



Specific strings pinpoint any difference between the two sets of strings

*if we index a reference **pan**genome, they pinpoint (mostly) all **novel** variations of a sample*

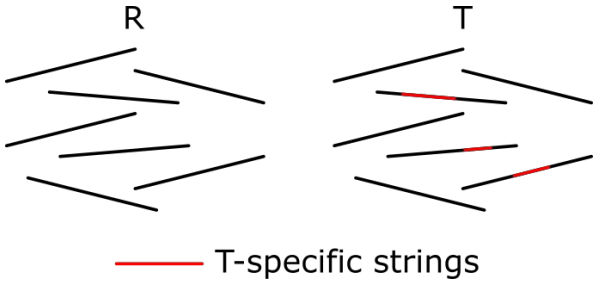
Why not using specific k-mers?*
Specific strings cover more variations (variable-length)

**P. Khorsand, L. Denti, et al. Bioinformatics Advances (2021)*

Pinpoint differences with specific strings

Given two sets of strings *(R,T)*, a **specific string** is any substring:

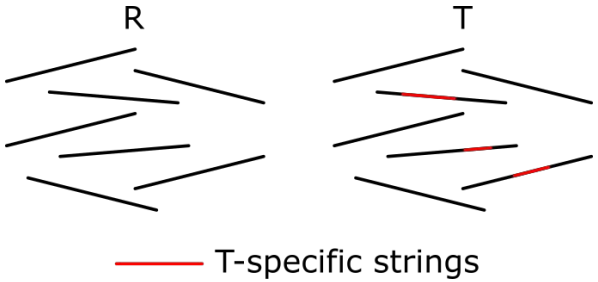
(i) occurring in T and not occurring in R *(substring)*

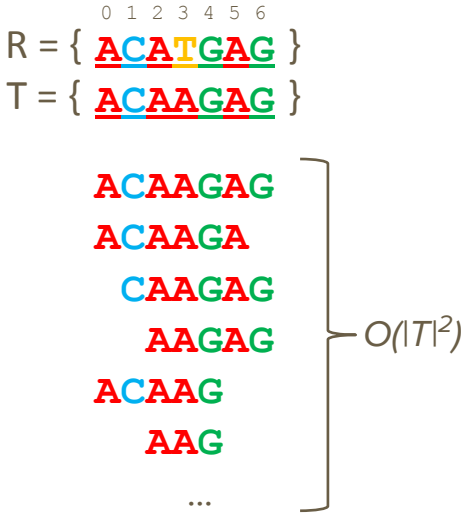


Pinpoint differences with specific strings

Given two sets of strings *(R,T)*, a **specific string** is any substring:

(i) occurring in T and not occurring in R *(substring)*



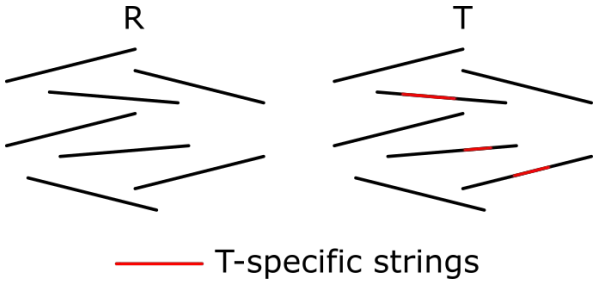


P. Khorsand, L. Denti, et al. ***Comparative genome analysis using sample-specific string detection in accurate long reads.*** *Bioinformatics Advances (2021)*

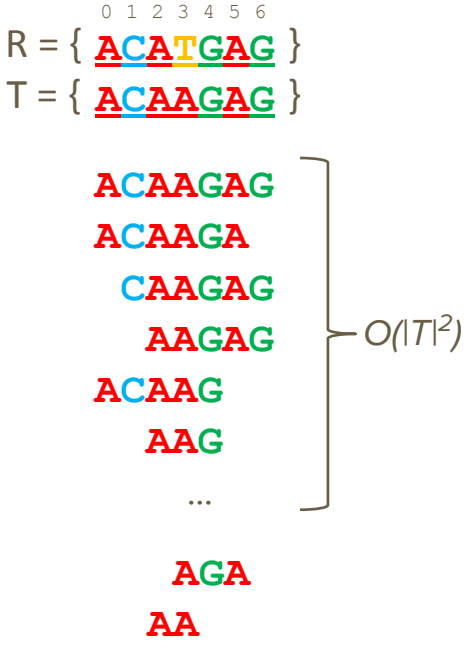
Pinpoint differences with specific strings

Given two sets of strings *(R,T)*, a **specific string** is any substring:

(i) occurring in T and not occurring in R *(substring)*



(ii) s.t. no other specific string is contained in it
(i.e., it is the shortest)

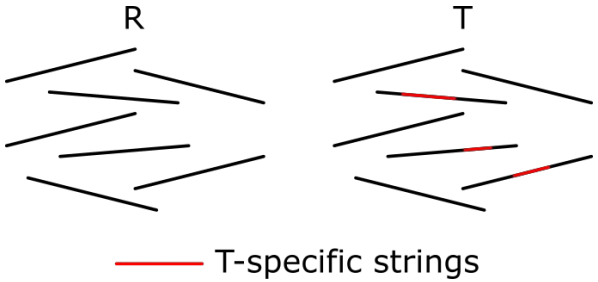


P. Khorsand, L. Denti, et al. ***Comparative genome analysis using sample-specific string detection in accurate long reads.*** *Bioinformatics Advances (2021)*

Pinpoint differences with specific strings

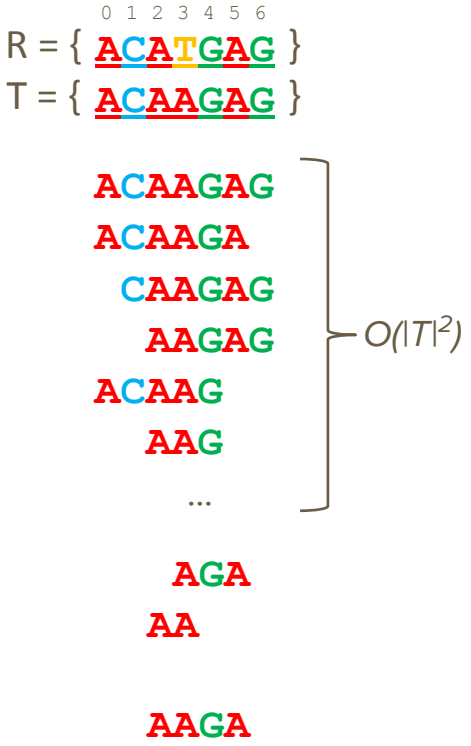
Given two sets of strings *(R,T)*, a **specific string** is any substring:

(i) occurring in T and not occurring in R *(substring)*



(ii) s.t. no other specific string is contained in it
(i.e., it is the shortest)

(super-)Specific: merge all specific strings overlapping on t∈T



P. Khorsand, L. Denti, et al. ***Comparative genome analysis using sample-specific string detection in accurate long reads.*** *Bioinformatics Advances (2021)*

Why specific strings?

Specific strings pinpoint any difference between two sets of strings

if we index a reference genome, they pinpoint (mostly) all variations

P. Khorsand, L. Denti, et al. ***Comparative genome analysis using sample-specific string detection in accurate long reads.*** *Bioinformatics Advances (2021)*

Why specific strings?

Specific strings pinpoint any difference between two sets of strings

*if we index a reference **pan**genome, they pinpoint (mostly) all **unknown/de novo** variations*

Why specific strings?

Specific strings pinpoint any difference between two sets of strings

*if we index a reference **pan**genome, they pinpoint (mostly) all **unknown/de novo** variations*

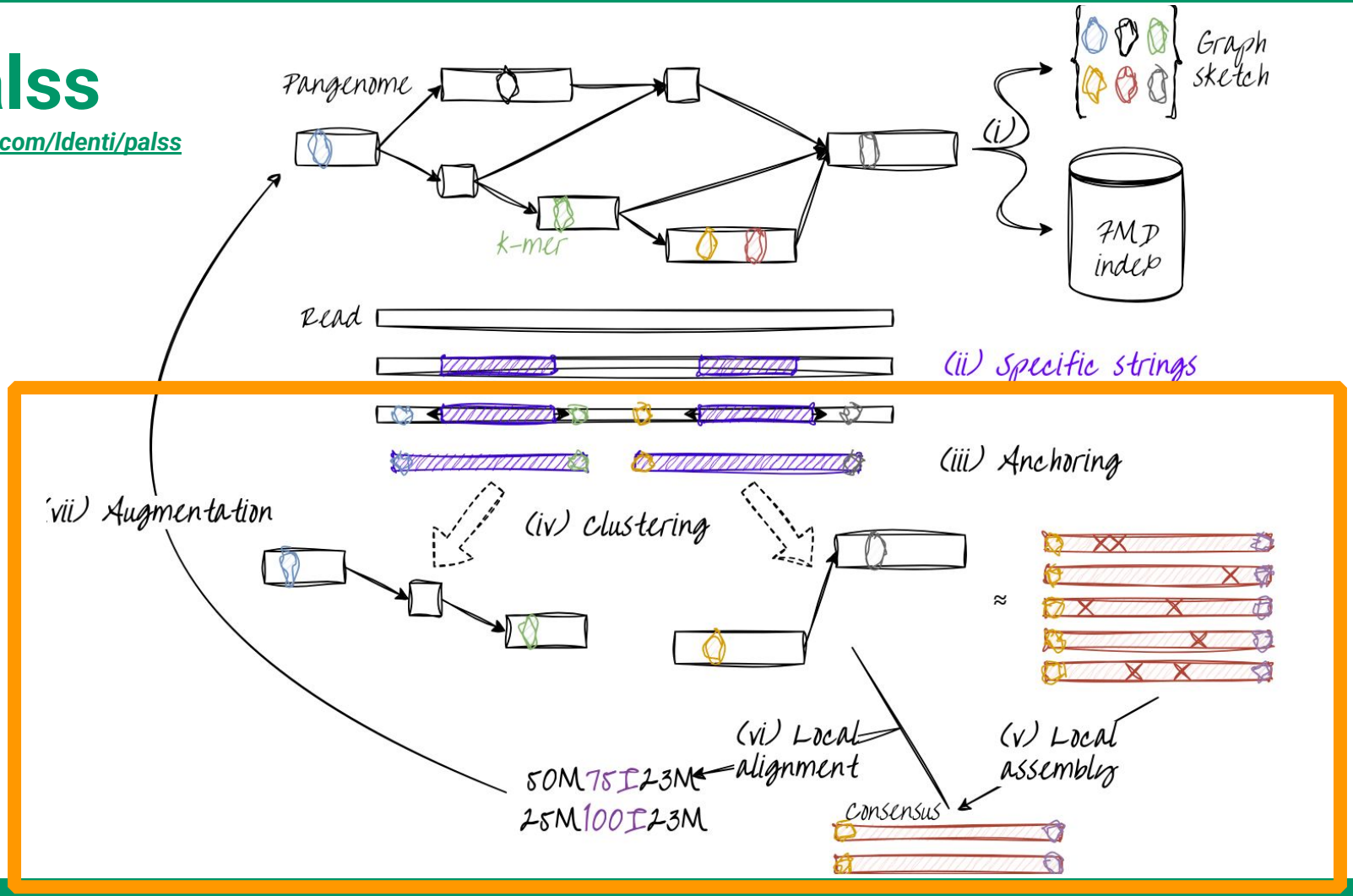
*Why not using specific k-mers?**

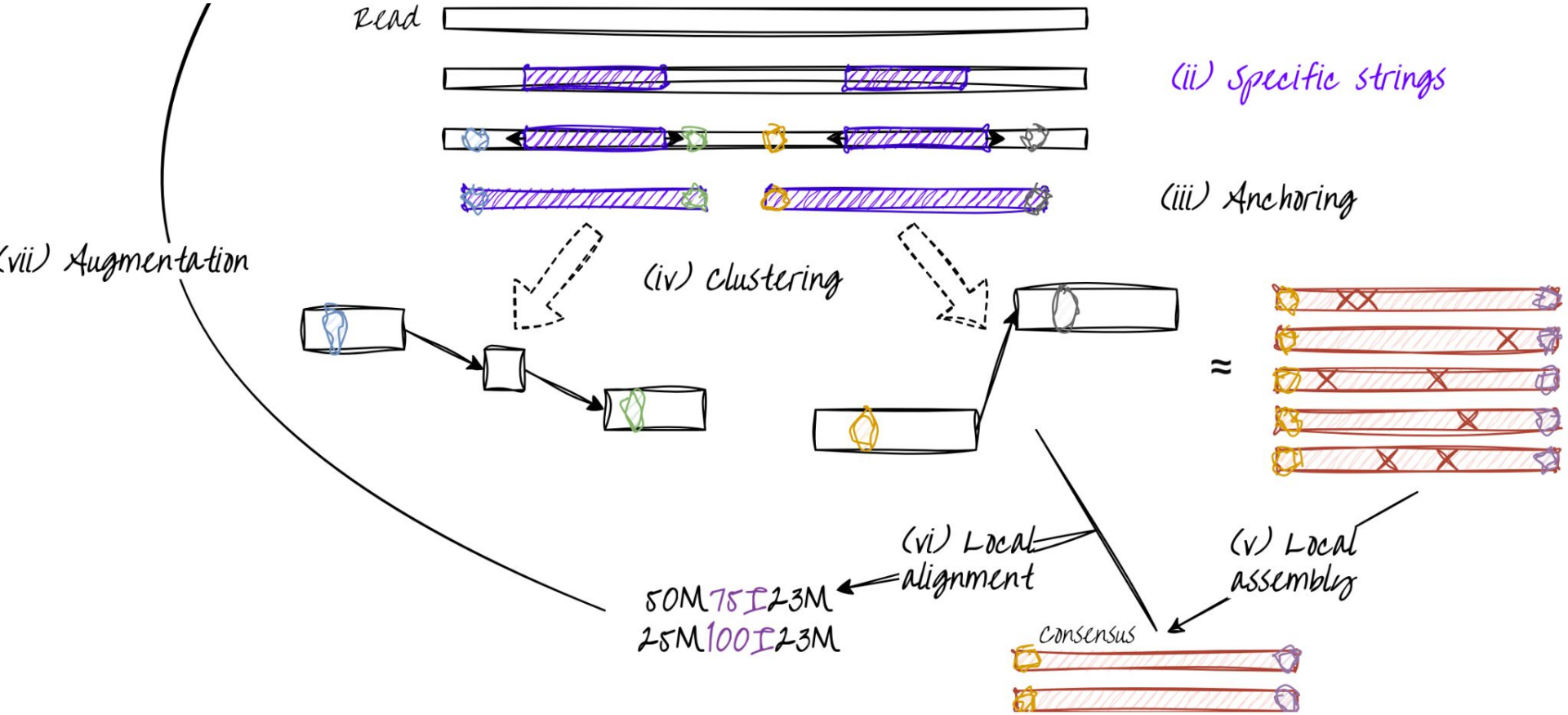
- *specific strings cover more variations (variable-length)*
- *(super-)specific strings are easier to compute*

P. Khorsand, L. Denti, et al. ***Comparative genome analysis using sample-specific string detection in accurate long reads.*** *Bioinformatics Advances (2021)*

palss

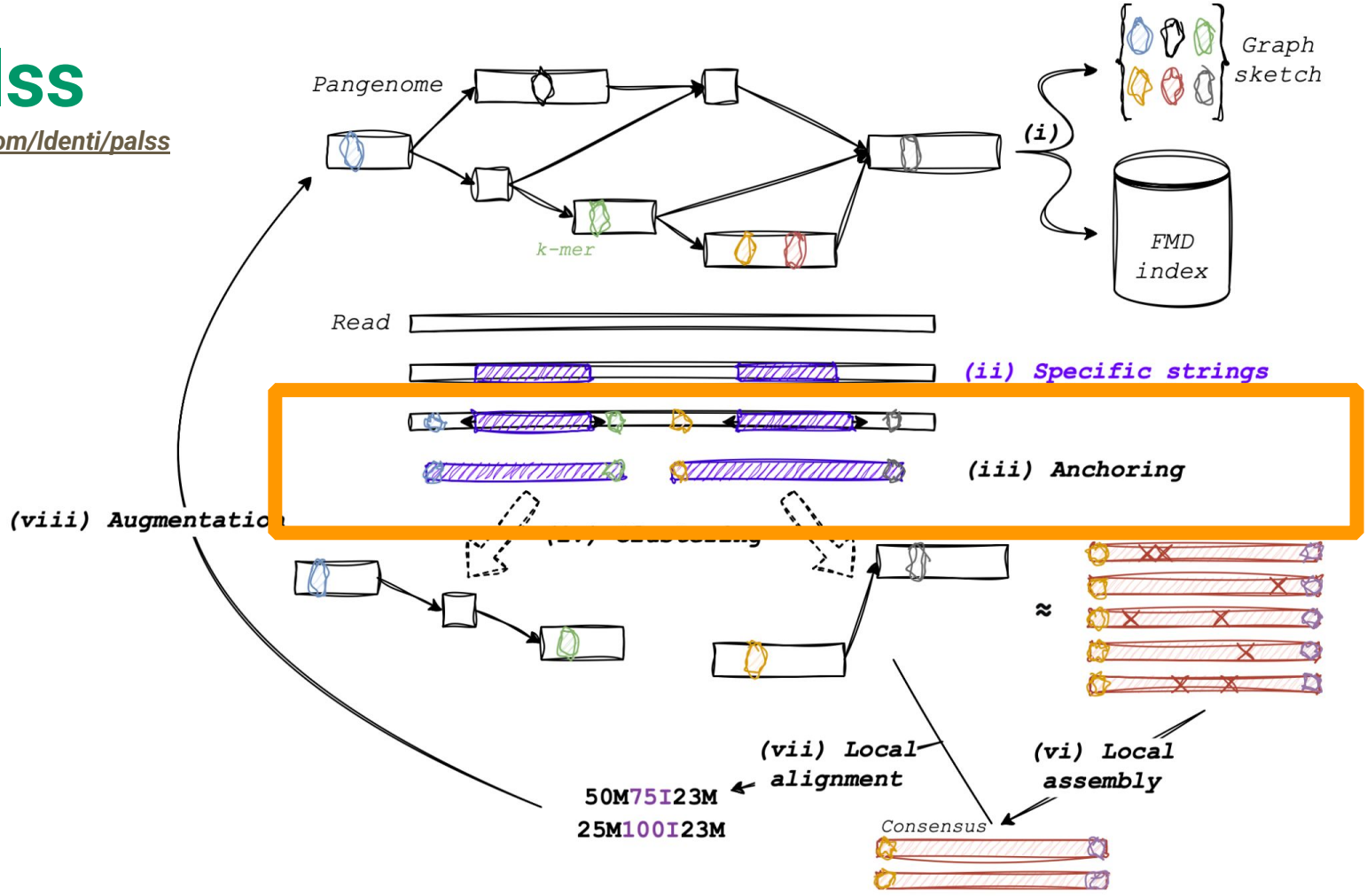
github.com/Identi/palss




Read
(ii) Specific strings
(iii) Anchoring
(vii) Augmentation
(iv) Clustering
≈
(vi) Local alignment
(v) Local assembly
50M75I23M
25M100I23M
consensus

palss

github.com/Identi/palss



Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)

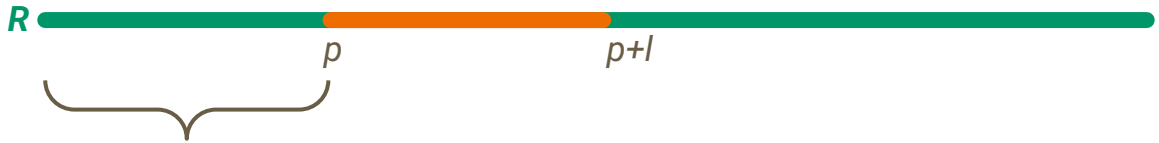
Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)

To anchor S to the graph, we search for solid anchors in its neighborhood:

Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)



To anchor S to the graph, we search for solid anchors in its neighborhood:

1. Select x(=20) solid anchors on the left of S *(i.e., in R[:p])*

Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)

To anchor S to the graph, we search for solid anchors in its neighborhood:

1. Select x(=20) solid anchors on the left of S *(i.e., in R[:p])*
2. Select x(=20) solid anchors on the right of S *(i.e., in R[p+l:])*

Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)

To anchor S to the graph, we search for solid anchors in its neighborhood:

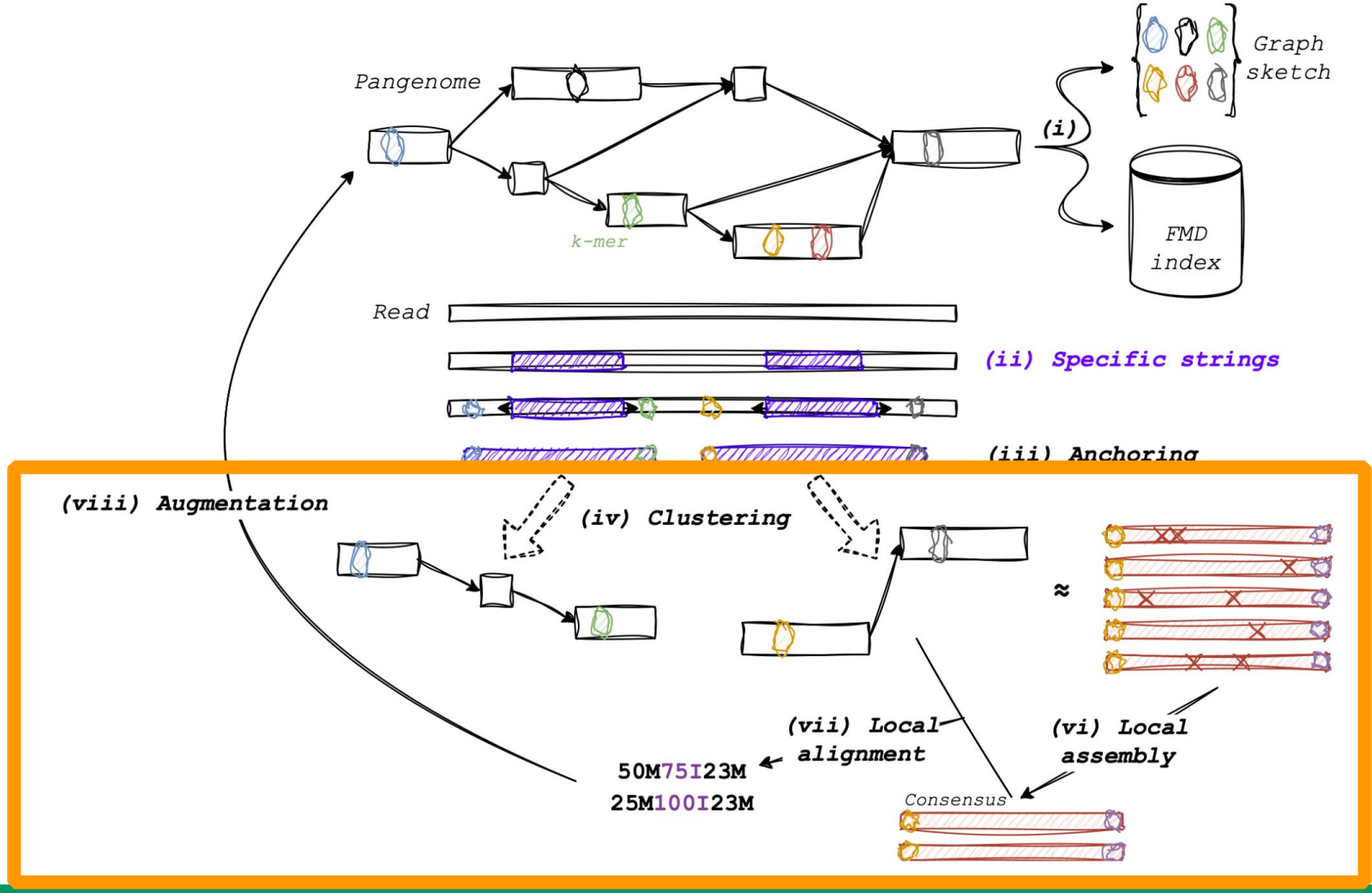
1. Select x(=20) solid anchors on the left of S *(i.e., in R[:p])*
2. Select x(=20) solid anchors on the right of S *(i.e., in R[p+l:])*
3. Pick the best pair *(closest on R and G, co-linear on at least one haplotype)*

Anchoring specific strings to graph *(WIP)*

A specific string **S** is a substring of a read **R** (S = *R[p:p+l]*)

To anchor S to the graph, we search for solid anchors in its neighborhood:

1. Select x(=20) solid anchors on the left of S *(i.e., in R[:p])*
2. Select x(=20) solid anchors on the right of S *(i.e., in R[p+l:])*
3. Pick the best pair *(closest on R and G, co-linear on at least one haplotype)*
4. Extend the specific string to include the two solid anchors


Pangenome
k-mer
Graph sketch
(i)
FMD index
Read
(ii) Specific strings
(iii) Anchoring
(viii) Augmentation
(iv) Clustering
≈
(vii) Local alignment
(vi) Local assembly
50M75I23M
25M100I23M
Consensus

Specific strings clustering

Each anchored specific string is now a tuple
(R, p, l, A, B), with A and B solid anchors

Specific strings clustering

Each anchored specific string is now a tuple
(R, p, l, A, B), with A and B solid anchors

Cluster them based on where they have been anchored

Specific strings clustering

Each anchored specific string is now a tuple
(R, p, l, A, B), with A and B solid anchors

Cluster them based on where they have been anchored (**sweep-line**):

1. Sort all specific strings by their left anchor (A)

Specific strings clustering

Each anchored specific string is now a tuple
(R, p, l, A, B), with A and B solid anchors

Cluster them based on where they have been anchored (**sweep-line**):

1. Sort all specific strings by their left anchor (A)
2. Create a cluster with the first specific string

Specific strings clustering

Each anchored specific string is now a tuple
(R, p, l, A, B), with A and B solid anchors

Cluster them based on where they have been anchored (**sweep-line**):

1. Sort all specific strings by their left anchor (A)
2. Create a cluster with the first specific string
3. Iterate over remaining strings while keeping one open cluster:
    - if new string overlap with current cluster, add it
    - otherwise, close current cluster and create a new one

Cluster analysis

Each cluster represents **specific portions** of the haplotypes of the new individual

We just need to summarize the information contained in each cluster to understand how to augment the graph

Cluster analysis

Each cluster represents **specific portions** of the haplotypes of the new individual

We just need to summarize the information contained in each cluster to understand how to augment the graph

Some challenges:

- specific strings may come from different haplotypes *(due to ploidy)*
- they may overlap and do not share anchors *(due to ploidy and sequencing errors)*

Cluster analysis

Each cluster represents **specific portions** of the haplotypes of the new individual

We just need to summarize the information contained in each cluster to understand how to augment the graph

Some challenges:

- specific strings may come from different haplotypes *(due to ploidy)*
- ~~they may overlap and do not share anchors *(due to ploidy and sequencing errors)*~~

Cluster analysis

A cluster is a triple *(A, B, **S**)*, *with left-most anchor A, right-most anchor B and anchored specific strings **S***

Cluster analysis

A cluster is a triple *(A, B, **S**)*, *with left-most anchor A, right-most anchor B and anchored specific strings **S***

1. Cluster is split based on specific strings length $\frac{\min(|s_1|, |s_2|)}{\max(|s_1|, |s_2|)} \geq r(=0.97)$
    - *heterozygous variations of different length*

Cluster analysis

A cluster is a triple *(A, B, **S**)*, *with left-most anchor A, right-most anchor B and anchored specific strings **S***

1. Cluster is split based on specific strings length
    - *heterozygous variations of different length*
2. Create one or two consensus per subcluster via POA (abpoa)
    - heterozygous variations of same size *(e.g., multi-allelic SNPs)*

Cluster analysis

A cluster is a triple *(A, B, **S**)*, *with left-most anchor A, right-most anchor B and anchored specific strings **S***

1. Cluster is split based on specific strings length
   - *heterozygous variations of different length*
2. Create one or two consensus per subcluster via POA (`abpoa`)
   - heterozygous variations of same size *(e.g., multi-allelic SNPs)*
3. Each consensus is locally aligned back to the graph (`ksw2`)
   - ✗ Actually, back to each “unique” haplotype path passing through the two anchors

Cluster analysis

A cluster is a triple *(A, B, **S**)*, *with left-most anchor A, right-most anchor B and anchored specific strings **S***

1. Cluster is split based on specific strings length
   - *heterozygous variations of different length*
2. Create one or two consensus per subcluster via POA (`abpoa`)
   - heterozygous variations of same size *(e.g., multi-allelic SNPs)*
3. Each consensus is locally aligned back to the graph (`ksw2`)
   - ✗ Actually, back to each “unique” haplotype path passing through the two anchors
4. Alignments are used to augment the graph (`vg augment`)