Minimizer-space de Bruijn graphs for pangenomics

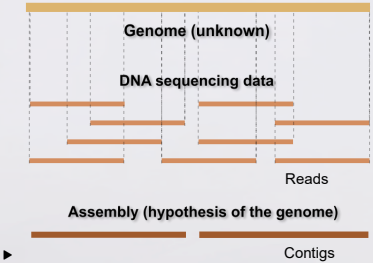
Rayan Chikhi

Institut Pasteur & CNRS

December 2021

44 years of assembling genomes

- **1977**: First complete genome assembled (phi X 174)
- **2003**: Human Genome Project completed
- **2014**: First $1,000 genome
- **2021**: Truly completed (Telomere-2-Telomere)



(Staden 1979) *"With modern fast sequencing techniques and suitable computer programs it is now possible to sequence whole genomes without the need of restriction maps."*

Long reads genome assembly

- Oxford Nanopore, PacBio CLR
  - 10-1,000 kbp reads, **5-12**% error rate
- PacBio HiFi
  - 10-25 kbp reads, $\leq$ **1**% error rate

Classical *de Bruijn* graphs not applicable (no long error-free *k*-mers). Instead:

- Overlap graphs (`Canu, miniasm, Shasta, Peregrine, ...`)
- Fuzzy dBGs (`wtdbg2`)
- Sparse dBGs: A-Bruijn or minimizers (`Flye, MBG`)

Challenge: Approaches don't scale (high resource usage, slow assembly time)!

Bacterial Pangenomics: Representing 100,000s bacterial genomes

- *k*-mer indexes (VARI, BiFrost, BlastFrost, ..)
- MinHash sketches (sourmash)
- Pangenome graphs?

Bacterial Pangenomics: Representing 100,000s bacterial genomes

- *k*-mer indexes (VARI, BiFrost, BlastFrost, ..)
- MinHash sketches (sourmash)
- Pangenome graphs?

Challenges:

- Terabyte-sized collections
- Graph construction
- Graph visualization

Bacterial Pangenomics: Representing 100,000s bacterial genomes

- *k*-mer indexes (VARI, BiFrost, BlastFrost, ..)
- MinHash sketches (sourmash)
- Pangenome graphs?

Challenges:

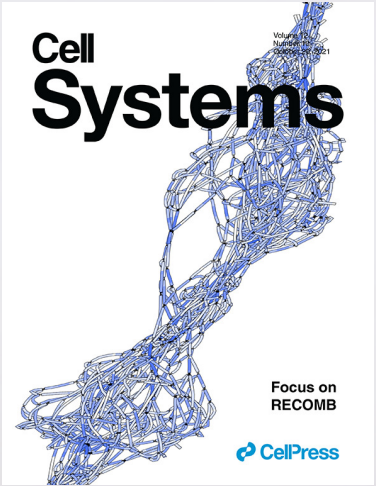
- Terabyte-sized collections
- Graph construction
- Graph visualization

In this talk: 100x-1000x cheaper pangenome graphs through controlled information loss

highly scalable dBGs: Minimizer-space de Bruijn graphs

Barış Ekim

Bonnie Berger



Preliminaries *k*-mers, de Bruijn graph (dBG)

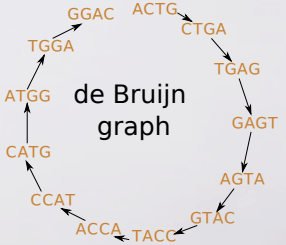
Reference genome: ACTGAGTACCATGGAC

Reads:
ACTGAGTAC
CTGAGTACCAT
GAGTACCATGGAC

Base-space

k-mers

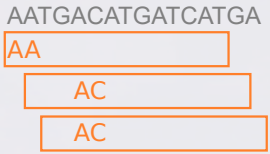
ACTG TACC GGAC
CTGA ACCA
TGAG CCAT
GAGT CATG
AGTA ATGG
GTAC TGGA



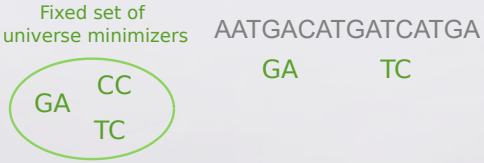
Preliminaries: Minimizers

Two kinds:

- **window**. Local: “smallest” *l*-mer in a window



- **universe**. Global: set of *l*-mers with low hash values



From now on: **universe**. (Also called Scaled MinHash)

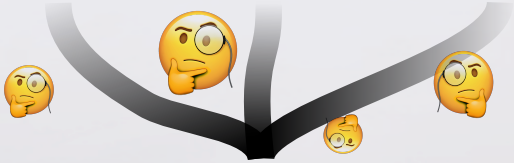
This work: stems from three ideas

Shasta
(Shafin et al, 2020)

Peregrine
(Chin et al, 2019)

index of minimizer pairs

wtdbg2
(Ruan et al, 2020)



mdBG

Our approach: Minimizers as *tokens* of the alphabet

Classical alphabet: $\Sigma_{DNA} = \{A, C, T, G\}$

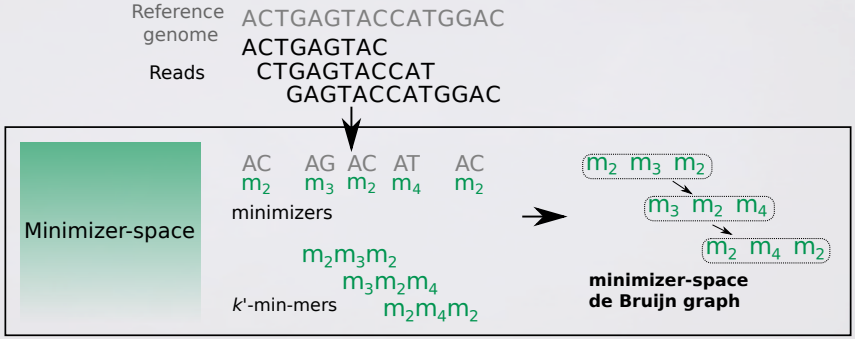
A k-mer with $k = 3$: AGT

Our approach: Minimizers as *tokens* of the alphabet

Classical alphabet: $\Sigma_{DNA} = \{A, C, T, G\}$
A k-mer with $k = 3$: AGT

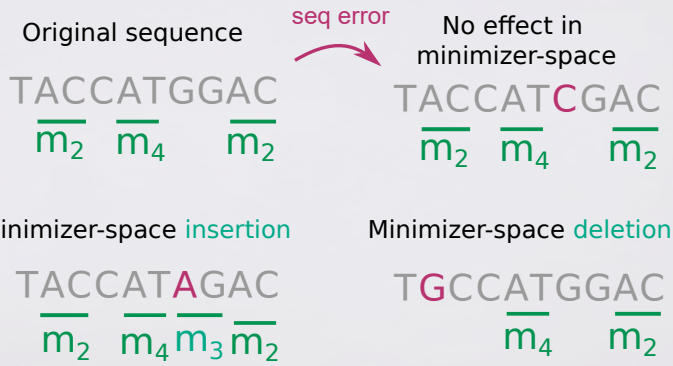
Minimizer alphabet: $\Sigma^{\ell} = \{\text{all minimizers of length } \ell\} = \{m_1, m_2, m_3, \ldots\}$
where e.g. $\ell = 2$, $m_1 = AA$, $m_2 = AC$, $m_3 = AG$, $m_2 = AT$
A k-mer over Σ^{ℓ} (a ***k*-min-mer**): $m_1 m_3 m_2$

Minimizer-space de Bruijn graph



A **minimizer-space de Bruijn graph** is a **de Bruijn graph** over the **minimizer alphabet**.
Nodes = k-min-mers,
Edges = exact overlaps between k-1 minimizers

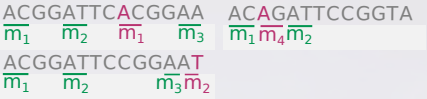
Sequencing errors propagate to minimizer-space



Error rate: base-space << minimizer-space,
e.g. 5% in base-space corresponds to 50% in minimizer-space.

Minimizer-space POA

(base-space POA: Lee *et al*, 2002)

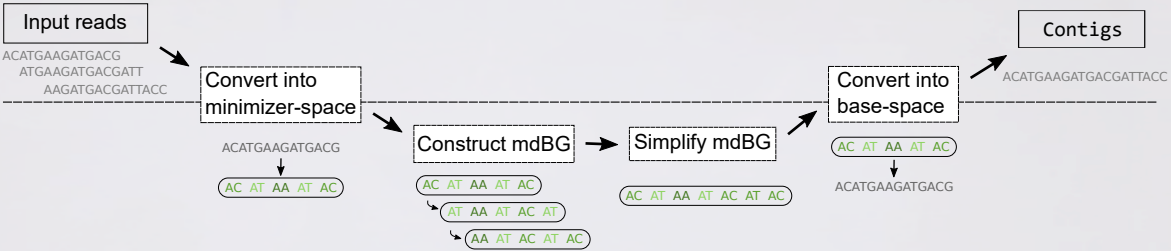


Final consensus

$m_1 m_2 m_3$

Applied to whole-genome *de novo* assembly

From accurate HiFi (< 1% error-rate) reads



Whole human PacBio HiFi (HG002) 50x coverage:

Tool name	`Peregrine`	`hifiasm`	`rust-mdbg`
Wall-clock time	14h8m	58h41m	**10m23s**
Memory usage	188 GB	195 GB	**10 GB**
# contigs	8109	431	805
NG50 (Mbp)	18.2	88.0	16.1
Genome fraction	97.0%	94.2%	95.5%

Assembly implementation details

- `gfatools` (H. Li, unpublished) for graph simplifications
- `2basespace` tool converts minimizer-space to base-space
- Automatic parameter setting (suboptimal):

$$\ell = 12$$

$$\text{density} = 0.003$$

$$k = \frac{3}{4} \cdot \text{avg\_readlen} \cdot \text{density}$$

- Multi-k script available (à la IDBA/SPAdes).
- Code available at `github.com/ekimb/rust-mdbg/`

Results: Metagenome assembly

Zymo D6331 mock metagenome HiFi

Species	Abundance	hifiasm	rust-mdbg
A. muciniphila	1.36%	100.000%	100.000%
B. fragilis	13.13%	99.994%	99.997%
B. adolescentis	1.34%	100.000%	99.730%
C. albican	1.61%	67.832%	39.821%
C. difficile	1.83%	99.996%	99.978%
C. perfringens	0.00%	0.005%	0.005%
E. faecalis	0.00%	0.006%	0.006%
E. coli B1109	8.44%	100.000%	97.918%
E. coli b2207	8.32%	100.000%	98.663%
E. coli B3008	8.25%	100.000%	99.558%
E. coli B766	7.83%	96.913%	96.270%

Species	Abundance	hifiasm	rust-mdbg
E. coli JM109	8.37%	100.000%	97.852%
F. prausnitzii	14.39%	100.000%	100.000%
F. nucleatum	3.78%	100.000%	99.960%
L. fermentum	0.86%	100.000%	100.000%
M. smithii	0.04%	99.840%	87.175%
P. corporis	5.37%	99.561%	99.561%
R. hominis	3.88%	100.000%	100.000%
S. cerevisiae	0.18%	69.522%	39.556%
S. enterica	0.02%	6.232%	4.619%
V. rogosae	11.02%	100.00%	100.000%

	hifiasm	rust-mdbg
Running time	34h29m	**55s**
Memory usage	83 GB	**0.9 GB**

Results: Pangenome graph of 661,405 bacterial genomes

Data from Blackwell et al, 2021:

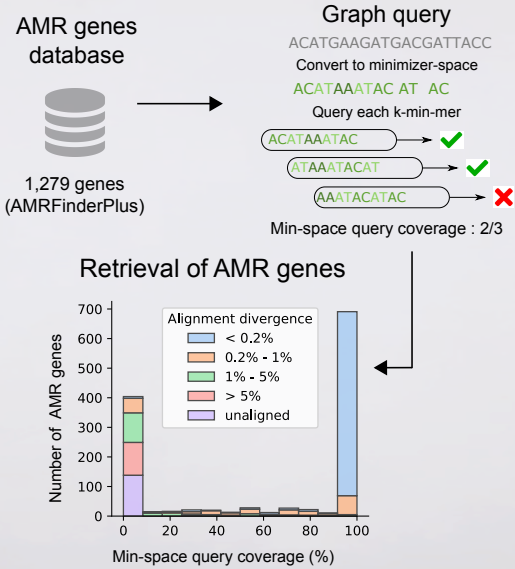
```
2.9T 661k_assemblies.fa
1.6T 661k_assemblies.fa.lz4
```

```
rust-mdbg -k 10 -l 12 --density 0.001 --minabund 1 661k_assemblies.fa.lz4
```

Largest 5 connected components:

Taxons in component	18	22	4	22	10
Dominant species	*Mycobacterium tuberculosis*	*Salmonella enterica*	*Burkholderia gladioli*	*Pseudomonas protegens*	*Cupriavidus alkaliphilus*

Biological results: Querying AMR genes



Behind the scenes of mdBG pangenome construction

- `rust-mdbg` software (see Barış' talk later today)
- set of scripts (`https://github.com/ekimb/rust-mdbg/tree/master/experiments/661k_genomes`)

In particular:

- Search for k-min-mers done with `grep`
- But fast: 10mins for a batch of queries
- k-min-mer compressed with `lz4` for fast decompression
- Graph stored without any sequence in GFA format (2-20GB `.gfa.gz`)
- "Resolution": 10kbp (kminmer span)

Behind the scenes of mdBG pangenome construction

- `rust-mdbg` software (see Barış' talk later today)
- set of scripts (`https://github.com/ekimb/rust-mdbg/tree/master/experiments/661k_genomes`)

In particular:

- Search for k-min-mers done with `grep`
- But fast: 10mins for a batch of queries
- k-min-mer compressed with `lz4` for fast decompression
- Graph stored without any sequence in GFA format (2-20GB `.gfa.gz`)
- "Resolution": 10kbp (kminmer span)

Potential hacking directions:

- Still non-trivial to separate connected components in a GFA
- Succinct (colored) mdBGs
- k-min-mer indexes
- visualization of pangenome mdBGs (100k-1M nodes)

FAQ on mdBGs

Short reads?

- Doesn't seem applicable

Different minimizers?

- In our limited experience, didn't play a big role

Improving assembly N50?

- Try better graph simplifications

Nanopore data?

- 5% error rate is too much, but 1% sounds promising

Conclusion

- **mdBGs** can not only perform genome assembly but also represent pangenome graphs for large collections (661k bacterial genomes) efficiently (10s of GB) at 10-100kbp resolution

Potential hacking directions (2):

- Higher-resolution mdBGs (1 kbp span for *k*-min-mers)
- Pangenome mdBGs for eukaryotes
- Automated differential analysis on colored GFAs
- Large structural variant calling on colored GFAs