k-mer data structures in sequence bioinformatics

Rayan Chikhi

Institut Pasteur & CNRS

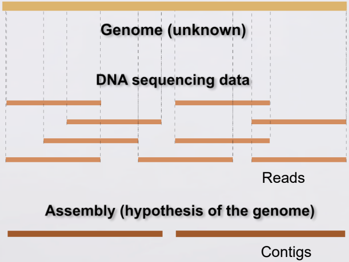
HDR defense, Sep 2021

1. A tale of optimizing the space usage of de Bruijn graphs
2. Minimizer-space de Bruijn graphs

M.	**KUCHEROV Gregory**	Rapporteur	DR,
Mme	**SCHBATH Sophie**	Rapportrice	DR,
M.	**STOYE Jens**	Rapporteur	Professeur,
Mme	**CARBONE Alessandra**	Examinatrice	Professeur,
M.	**GASCUEL Olivier**	Examinateur	DR,
M.	**RICHARD Hugues**	Examinateur	MCF,
M.	**VALLENET David**	Examinateur	DR,
M.	**VINAR Tomas**	Examinateur	Professeur

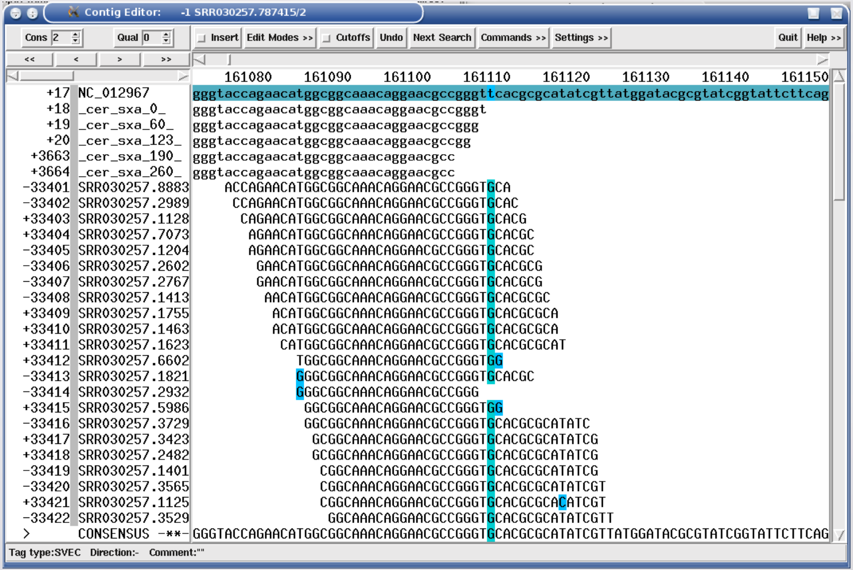
44 years of genome assembly

- **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.”*

Algorithmic pre-history



```
                          161080    161090    161100    161110    161120    161130    161140    161150
   +17 NC_012967      gggtaccagaacatggcggcaaacaggaacgccgggttcacgcgcatatcgttatggatacgcgtatcggtattcttcag
   +18 _cer_sxa_0_    gggtaccagaacatggcggcaaacaggaacgccgggt
   +19 _cer_sxa_60_   gggtaccagaacatggcggcaaacaggaacgccggg
   +20 _cer_sxa_123_  gggtaccagaacatggcggcaaacaggaacgccgg
 +3663 _cer_sxa_190_  gggtaccagaacatggcggcaaacaggaacgcc
 +3664 _cer_sxa_260_  gggtaccagaacatggcggcaaacaggaacgcc
-33401 SRR030257.8883     ACCAGAACATGGCGGCAAACAGGAACGCCGGGTGCA
-33402 SRR030257.2989      CCAGAACATGGCGGCAAACAGGAACGCCGGGTGCAC
+33403 SRR030257.1128       CAGAACATGGCGGCAAACAGGAACGCCGGGTGCACG
+33404 SRR030257.7073        AGAACATGGCGGCAAACAGGAACGCCGGGTGCACGC
-33405 SRR030257.1204        AGAACATGGCGGCAAACAGGAACGCCGGGTGCACGC
-33406 SRR030257.2602         GAACATGGCGGCAAACAGGAACGCCGGGTGCACGCG
-33407 SRR030257.2767         GAACATGGCGGCAAACAGGAACGCCGGGTGCACGCG
-33408 SRR030257.1413          AACATGGCGGCAAACAGGAACGCCGGGTGCACGCGC
+33409 SRR030257.1755           ACATGGCGGCAAACAGGAACGCCGGGTGCACGCGCA
+33410 SRR030257.1463           ACATGGCGGCAAACAGGAACGCCGGGTGCACGCGCA
+33411 SRR030257.1623            CATGGCGGCAAACAGGAACGCCGGGTGCACGCGCAT
+33412 SRR030257.6602              TGGCGGCAAACAGGAACGCCGGGTGG
-33413 SRR030257.1821              GGGCGGCAAACAGGAACGCCGGGTGCACGC
-33414 SRR030257.2932              GGGCGGCAAACAGGAACGCCGGG
+33415 SRR030257.5986               GGCGGCAAACAGGAACGCCGGGTGG
-33416 SRR030257.3729               GGCGGCAAACAGGAACGCCGGGTGCACGCGCATATC
+33417 SRR030257.3423                GCGGCAAACAGGAACGCCGGGTGCACGCGCATATCG
+33418 SRR030257.2482                GCGGCAAACAGGAACGCCGGGTGCACGCGCATATCG
-33419 SRR030257.1401                 CGGCAAACAGGAACGCCGGGTGCACGCGCATATCG
-33420 SRR030257.3565                 CGGCAAACAGGAACGCCGGGTGCACGCGCATATCGT
+33421 SRR030257.1125                 CGGCAAACAGGAACGCCGGGTGCACGCGCACATCGT
-33422 SRR030257.3529                  GGCAAACAGGAACGCCGGGTGCACGCGCATATCGTT
     > CONSENSUS -**- GGGTACCAGAACATGGCGGCAAACAGGAACGCCGGGTGCACGCGCATATCGTTATGGATACGCGTATCGGTATTCTTCAG
```

Tag type:SVEC Direction:- Comment:""

Screenshot: MIRA

Algorithmic pre-history

1. **Assembly using strings**
   - Shortest Common Superstring (Kececioglu, Myers 1993)
   - Greedy algorithms (CAP3 from Huang, Madan 1999)

Algorithmic pre-history

1. **Assembly using strings**
   - Shortest Common Superstring (Kececioglu, Myers 1993)
   - Greedy algorithms (CAP3 from Huang, Madan 1999)
2. **Assembly using graphs**: string graphs and de Bruijn Graphs (both from DIMACS'94)

A History of DNA Sequence Assembly, G. Myers, 2016

dBGs widely used across genomics (SPAdes: 13,000 citations; Trinity: 12,000 citations)

Algorithmic pre-history

1. **Assembly using strings**
   - Shortest Common Superstring (Kececioglu, Myers 1993)
   - Greedy algorithms (CAP3 from Huang, Madan 1999)
2. **Assembly using graphs**: string graphs and de Bruijn Graphs (both from DIMACS’94)

A History of DNA Sequence Assembly, G. Myers, 2016

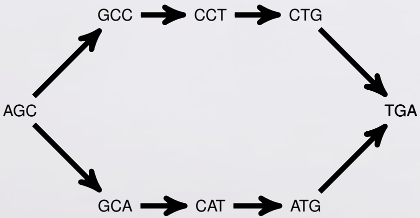
dBGs widely used across genomics (SPAdes: 13,000 citations; Trinity: 12,000 citations)

de Bruijn graph

A **de Bruijn** graph for a fixed integer k:

1. **Nodes** = all k*-mers* (substrings of length k) in the reads
2. **Edges** = all exact overlaps of length exactly $(k-1)$

dBG, $k = 3$:



Of those reads:

AGCCTGA

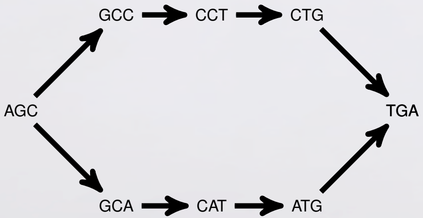
AGCATGA

de Bruijn graph

A **de Bruijn** graph for a fixed integer k:

1. **Nodes** = all k-*mers* (substrings of length k) in the reads
2. **Edges** = all exact overlaps of length exactly $(k-1)$

dBG, $k = 3$:



Of those reads:

AGCCTGA

AGCATGA

dBG of *E. coli* reads, k=71:

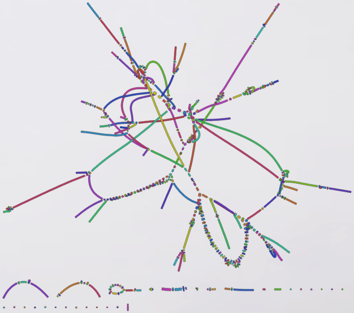
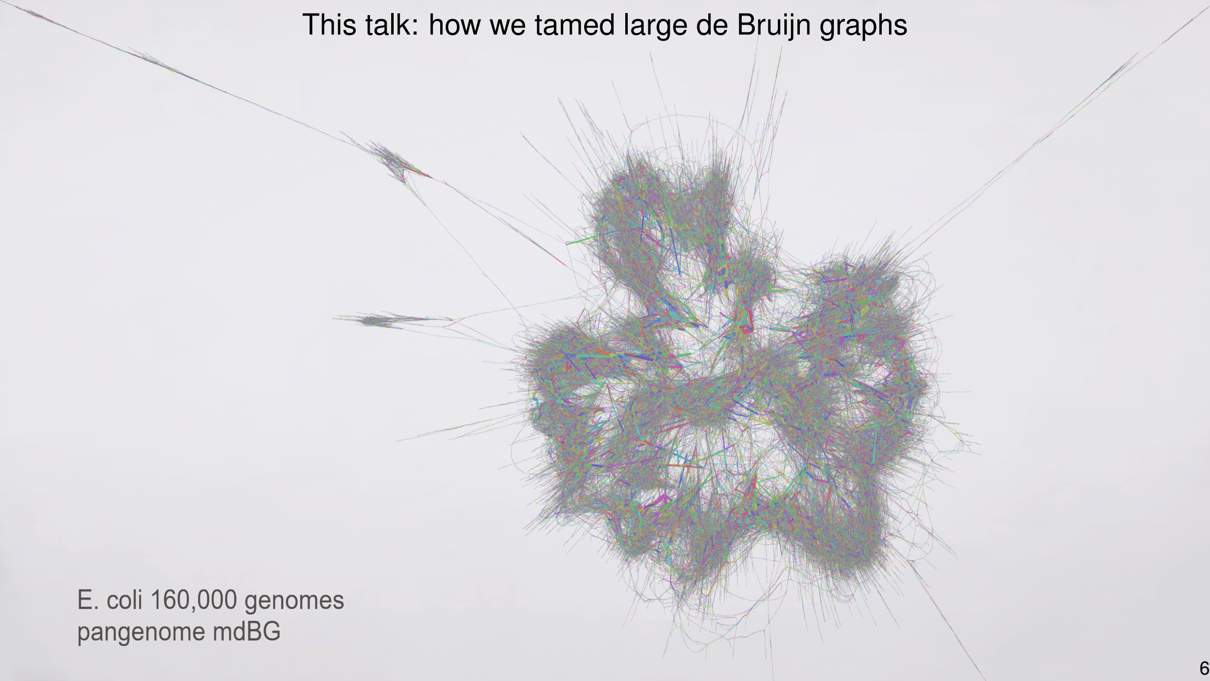


Fig: Bandage

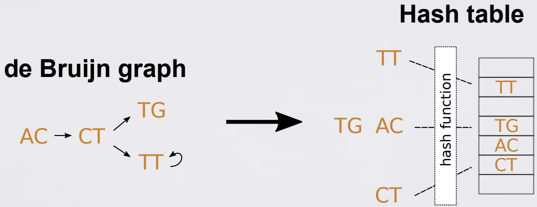
This talk: how we tamed large de Bruijn graphs



E. coli 160,000 genomes
pangenome mdBG

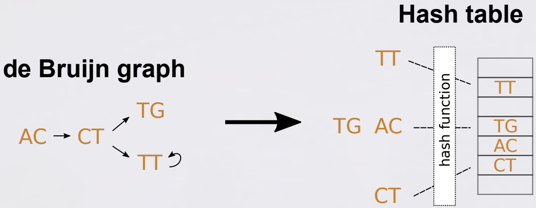
The early days (2008-2010)

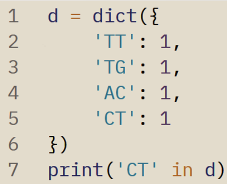
- Short-read genome assemblers (EULER-SR, Velvet, SOAPdenovo, ABySS)
- **Limited by machine memory** (Most efficient: SOAPdenovo, 120 GB for human)



The early days (2008-2010)

- Short-read genome assemblers (EULER-SR, Velvet, SOAPdenovo, ABySS)
- **Limited by machine memory** (Most efficient: SOAPdenovo, 120 GB for human)

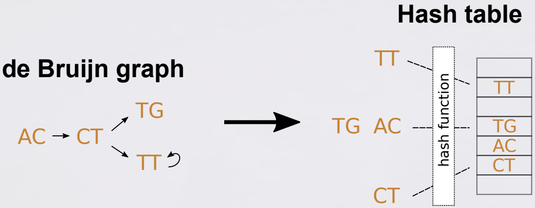


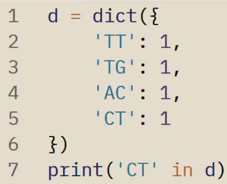


```
d = dict({
      'TT': 1,
      'TG': 1,
      'AC': 1,
      'CT': 1
})
print('CT' in d)
```

The early days (2008-2010)

- Short-read genome assemblers (EULER-SR, Velvet, SOAPdenovo, ABySS)
- **Limited by machine memory** (Most efficient: SOAPdenovo, 120 GB for human)





```
d = dict({
    'TT': 1,
    'TG': 1,
    'AC': 1,
    'CT': 1
})
print('CT' in d)
```

Low contiguity though:

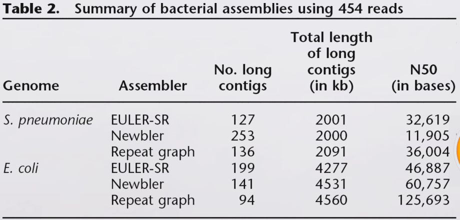


Table 2. Summary of bacterial assemblies using 454 reads

Genome	Assembler	No. long contigs	Total length of long contigs (in kb)	N50 (in bases)
S. pneumoniae	EULER-SR	127	2001	32,619
	Newbler	253	2000	11,905
	Repeat graph	136	2091	36,004
E. coli	EULER-SR	199	4277	46,887
	Newbler	141	4531	60,757
	Repeat graph	94	4560	125,693

Table from Chaisson *et al* 2008

The birth of a line of research (2011)

- Conway & Bromage (2011)
- **Assembly graphs can actually be stored efficiently**

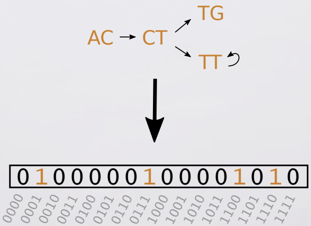
The birth of a line of research (2011)

- Conway & Bromage (2011)
- **Assembly graphs can actually be stored efficiently**
- Create a large array of 4^k positions (e.g. 4^{20} is a terabit)
- Put **1**s at positions of *k*-mers
- Can be **compressed optimally** while supporting queries (Okanohara *et al* 2006)

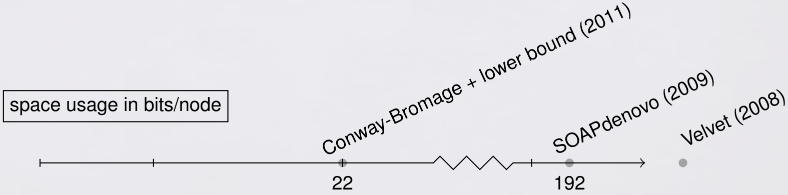
de Bruijn graph

A=00 G=10
C=01 T=11

AC=0001 TG=1110
CT=0111 TA=1100

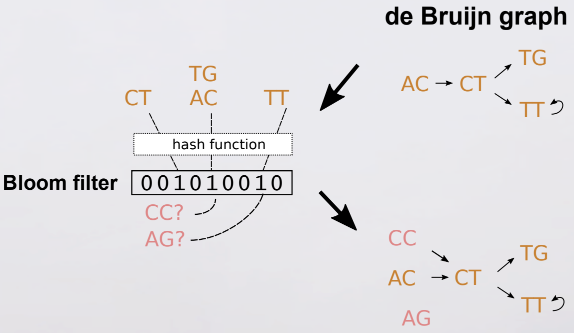


Is this the end of the research line?



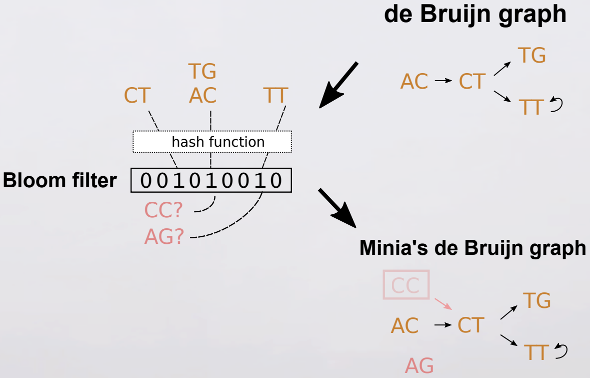
Beating the lower bound (by inexactness, 2012)

- **2011**: Pell *et al* proposed an inexact dBG representation.
- Bit vector is replaced by a Bloom filter.



Beating the lower bound (by inexactness, 2012)

- **2011**: Pell *et al* proposed an inexact dBG representation.
- Bit vector is replaced by a Bloom filter.
- **2012**: G. Rizk & I proposed **Minia**
- Same idea, but made the graph exact *where it matters*

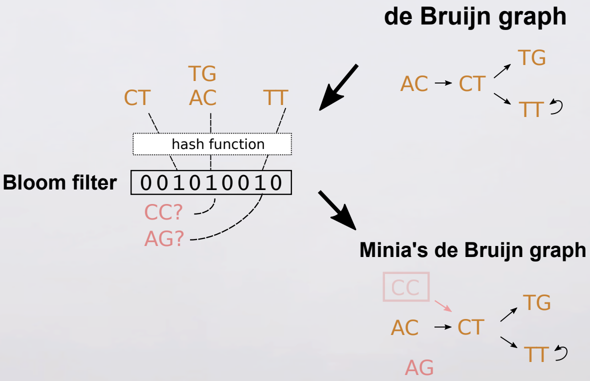


Guillaume Rizk

PhD self

Beating the lower bound (by inexactness, 2012)

- **2011**: Pell *et al* proposed an inexact dBG representation.
- Bit vector is replaced by a Bloom filter.
- **2012**: G. Rizk & I proposed **Minia**
- Same idea, but made the graph exact *where it matters*

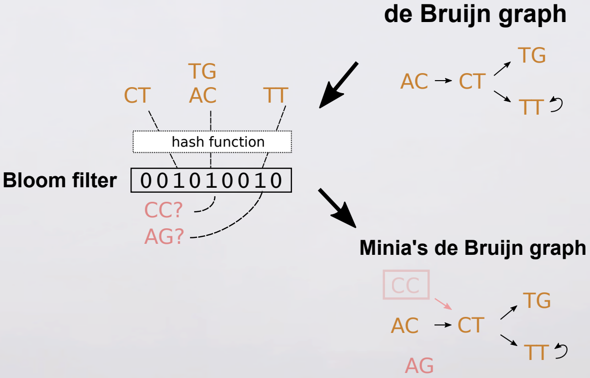


Guillaume Rizk

PhD self

Beating the lower bound (by inexactness, 2012)

- **2011**: Pell *et al* proposed an inexact dBG representation.
- Bit vector is replaced by a Bloom filter.
- **2012**: G. Rizk & I proposed **Minia**
- Same idea, but made the graph exact *where it matters*
- **Small space!** Beats bit vectors by 2x.



Guillaume Rizk

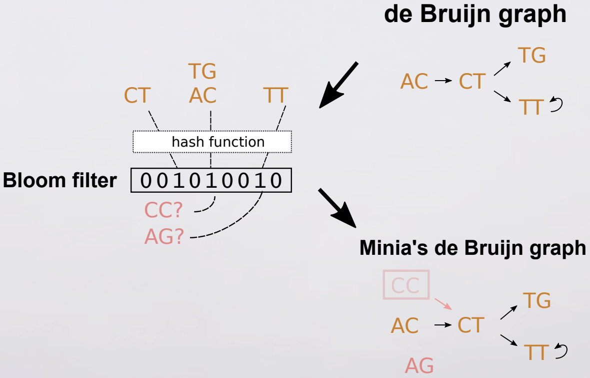
PhD self

Beating the lower bound (by inexactness, 2012)

- **2011**: Pell *et al* proposed an inexact dBG representation.
- Bit vector is replaced by a Bloom filter.
- **2012**: G. Rizk & I proposed **Minia**
- Same idea, but made the graph exact *where it matters*
- **Small space!** Beats bit vectors by 2x.
- First assembly of a human genome on a desktop computer.

Guillaume Rizk

PhD self



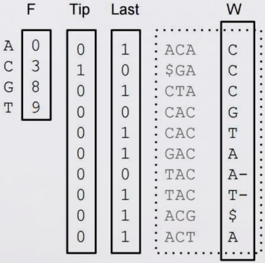
de Bruijn graph

Bloom filter

Minia's de Bruijn graph

Beating the lower bound (by instance specificity, 2012)

- **2012:** Sadakane *et al* proposed the **BOSS** encoding.
- Burrows-Wheeler transform modified to store a set of *k*-mers.



	F
A	0
C	3
G	8
T	9

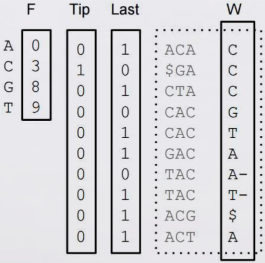
Tip	Last		W
0	1	ACA	C
1	0	$GA	C
0	1	CTA	C
0	0	CAC	G
0	1	CAC	T
0	1	GAC	A
0	0	TAC	A-
0	1	TAC	T-
0	1	ACG	$
0	1	ACT	A

Fig: MEGAHIT

Fun fact: Minia and BOSS were both introduced at WABI’12

Beating the lower bound (by instance specificity, 2012)

- **2012:** Sadakane *et al* proposed the **BOSS** encoding.
- Burrows-Wheeler transform modified to store a set of *k*-mers.
- **Very small space!** even smaller than Minia.



	F
A	0
C	3
G	8
T	9

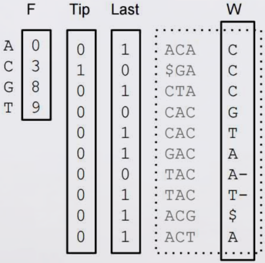
Tip	Last		W
0	1	ACA	C
1	0	$GA	C
0	1	CTA	C
0	0	CAC	G
0	1	CAC	T
0	1	GAC	A
0	0	TAC	A-
0	1	TAC	T-
0	1	ACG	$
0	1	ACT	A

Fig: MEGAHIT

Fun fact: Minia and BOSS were both introduced at WABI'12

Beating the lower bound (by instance specificity, 2012)

- **2012:** Sadakane *et al* proposed the **BOSS** encoding.
- Burrows-Wheeler transform modified to store a set of *k*-mers.
- **Very small space!** even smaller than Minia.
- But, some limitations (reverse complements, & took years to implement)



	F
A	0
C	3
G	8
T	9

Tip	Last		W
0	1	ACA	C
1	0	$GA	C
0	1	CTA	C
0	0	CAC	G
0	1	CAC	T
0	1	GAC	A
0	0	TAC	A-
0	1	TAC	T-
0	1	ACG	$
0	1	ACT	A

Fig: MEGAHIT

Fun fact: Minia and BOSS were both introduced at WABI'12

New perspective on the topic (2014)

- So, how comes Minia & BOSS beat the lower bound?

[1] R Chikhi, A Limasset, S Jackman, JT Simpson, P Medvedev, *On the representation of de Bruijn graphs*, RECOMB'14

New perspective on the topic (2014)

- So, how comes Minia & BOSS beat the lower bound?
- The lower bound assumed the graph was **exact**.
- Minia only supports **some** operations exactly.

[1] R Chikhi, A Limasset, S Jackman, JT Simpson, P Medvedev, *On the representation of de Bruijn graphs*, RECOMB’14

New perspective on the topic (2014)

Paul Medvedev

Antoine Limasset

Postdoc self

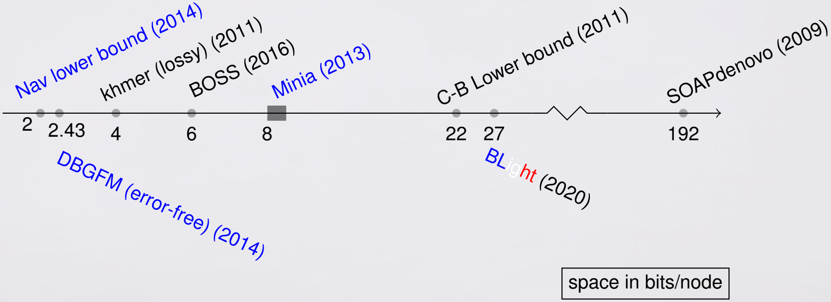
- So, how comes Minia & BOSS beat the lower bound?
- The lower bound assumed the graph was **exact**.
- Minia only supports **some** operations exactly.

- We came up with new lower bounds, i.e. ≈3 bits instead of 22 bits.[1]
- Open problem: a matching upper bound in the general case
- *Intriguingly fun fact*[2]: BOSS is fully exact (same as bit vector) and yet still beats the lower bound

[1] R Chikhi, A Limasset, S Jackman, JT Simpson, P Medvedev, *On the representation of de Bruijn graphs*, RECOMB'14

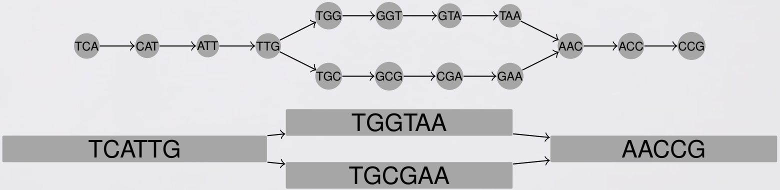
[2] For the handful of people on Earth who find this fun

Where are we now? (& my contribs)



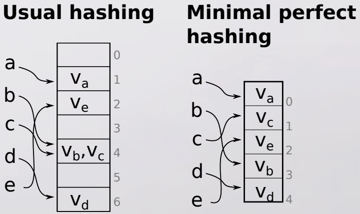
Current works on dBGs (2016-2020)

k-mer counting	Compaction	MPHF based	FM-Index based	General purpose
KMC3	BCALM2*	Pufferfish	DBGFM*	BBHash*
DSK2*	Cuttlefish	BLight	BOSS	Bifrost
Jellyfish2	TwoPaCo	FDBG*	dynamicBOSS	
SPAdes-kmercounter			bufBOSS	



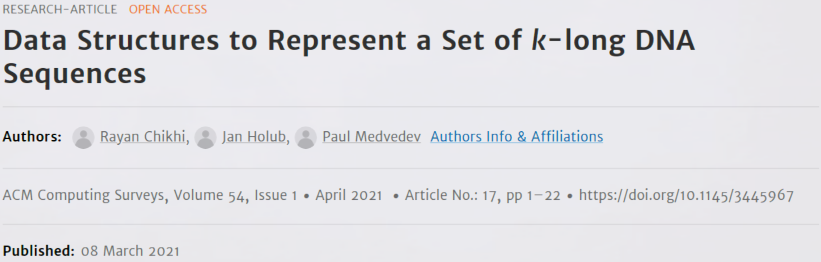
Nowadays:

- Focus is less on space, more on features
- Fast query times, associativity, dynamicity


Usual hashing

Minimal perfect hashing

A more complete review



RESEARCH-ARTICLE OPEN ACCESS

Data Structures to Represent a Set of *k*-long DNA Sequences

Authors: Rayan Chikhi, Jan Holub, Paul Medvedev Authors Info & Affiliations

ACM Computing Surveys, Volume 54, Issue 1 • April 2021 • Article No.: 17, pp 1–22 • https://doi.org/10.1145/3445967

Published: 08 March 2021

Part 2

Long reads genome assembly

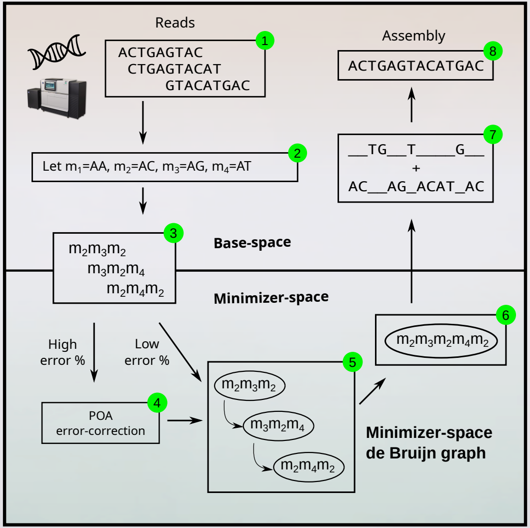
- Oxford Nanopore, PacBio CLR
    - 10-1,000 kbp reads, **5-12**% error rate
- PacBio HiFi
    - 10-25 kbp reads, $\leq$ **1**% error rate

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)!

dBGs on long reads: Minimizer-space de Bruijn graphs



Reads

1

ACTGAGTAC
CTGAGTACAT
GTACATGAC

2

Let m_1=AA, m_2=AC, m_3=AG, m_4=AT

3

$m_2m_3m_2$
$m_3m_2m_4$
$m_2m_4m_2$

Base-space

Minimizer-space

High error %

Low error %

4

POA
error-correction

5

$m_2m_3m_2$
$m_3m_2m_4$
$m_2m_4m_2$

Minimizer-space de Bruijn graph

6

$m_2m_3m_2m_4m_2$

7

___TG___T_____G___
+
AC___AG_ACAT_AC

Assembly

8

ACTGAGTACATGAC

- Long read human genome assembly on a desktop computer

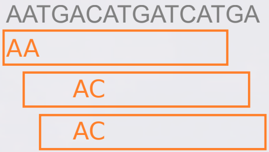
Barış Ekim

Bonnie Berger

Preliminaries: Minimizers

Two kinds:

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



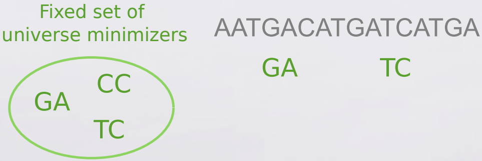
AATGACATGATCATGA

AA

AC

AC

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



Fixed set of universe minimizers

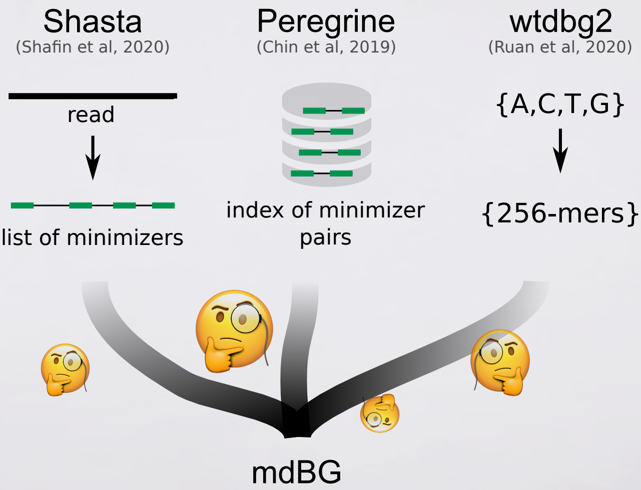
GA CC TC

AATGACATGATCATGA

GA TC

From now on: **universe**.

This work: stems from three ideas



Shasta
(Shafin et al, 2020)

read

list of minimizers

Peregrine
(Chin et al, 2019)

index of minimizer pairs

wtdbg2
(Ruan et al, 2020)

{A,C,T,G}

{256-mers}

mdBG

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:

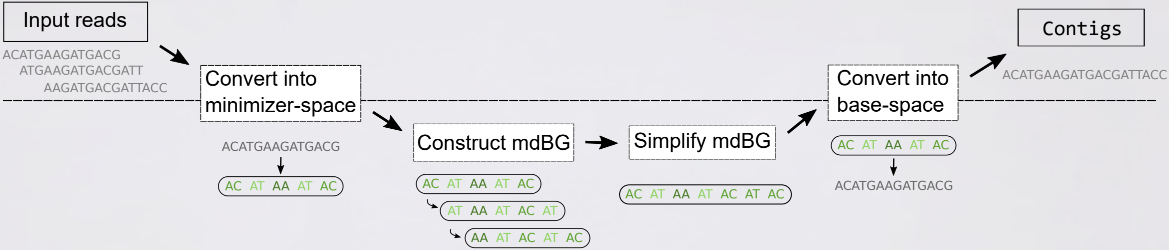
$$\Sigma^{\ell} = \{\text{all minimizers of length } \ell\} = \{m_1, m_2, m_3, \ldots\}$$

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**): $m_1 m_3 m_2$

Results: Whole-genome *de novo* assembly

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



Input reads

ACATGAAGATGACG
ATGAAGATGACGATT
AAGATGACGATTACC

Convert into minimizer-space

ACATGAAGATGACG

AC AT AA AT AC

Construct mdBG

AC AT AA AT AC
AT AA AT AC AT
AA AT AC AT AC

Simplify mdBG

AC AT AA AT AC AT AC

Convert into base-space

AC AT AA AT AC

ACATGAAGATGACG

Contigs

ACATGAAGATGACGATTACC

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%

Results: Metagenome assembly

Zymo D6331 mock metagenome HiFi

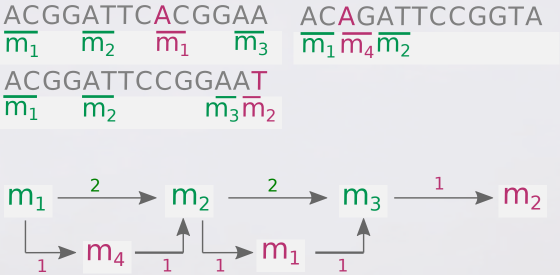
Species	**Abundance**	`hifiasm-m`	`rust-mdbg`
A. muciniphila	1.36%	100.000%	100.000%
B. fragilis	13.13%	99.994%	99.997%
B. adolescentis	1.34%	100.000%	99.730%
C. albican	1.61%	67.832%	39.821%
C. difficile	1.83%	99.996%	99.978%
C. perfringens	0.00%	0.005%	0.005%
E. faecalis	0.00%	0.006%	0.006%
E. coli 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-m`	`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-m`	`rust-mdbg`
Running time	34h29m	**55s**
Memory usage	83 GB	**0.9 GB**

For > 1% error rates: Minimizer-space POA error correction

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



ACGGATTCACGGAA

m_1 m_2 m_1 m_3

ACAGATTCCGGTA

m_1 m_4 m_2

ACGGATTCCGGAAT

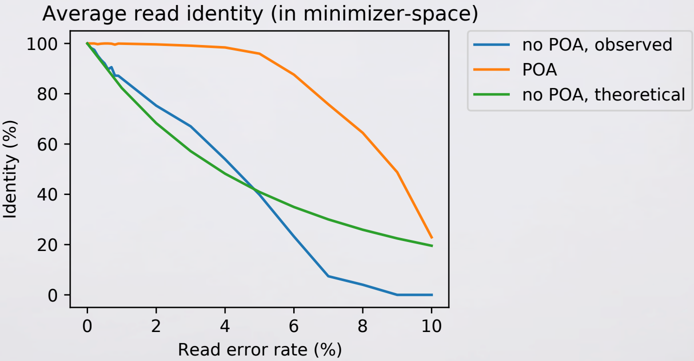
m_1 m_2 m_3 m_2

Final consensus

$m_1 m_2 m_3$

For > 1% error rates: Minimizer-space POA error correction

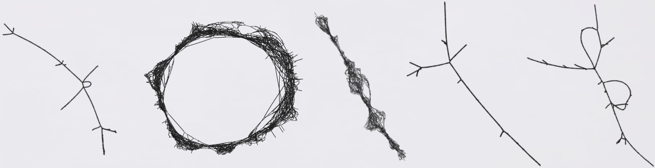
(base-space POA: Lee *et al*, 2002)



So, not quite ready for Nanopore data (≥ 5%).

Results: Pangenome graph of 661,405 bacterial genomes

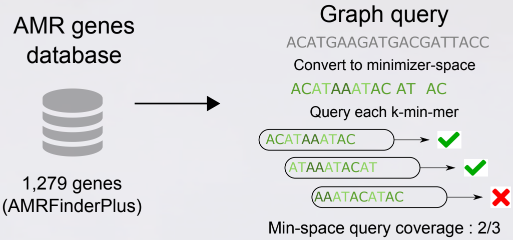
Largest 5 connected components:



+ 725,820 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



AMR genes database

1,279 genes (AMRFinderPlus)

Graph query

ACATGAAGATGACGATTACC

Convert to minimizer-space

ACATAAATAC AT AC

Query each k-min-mer

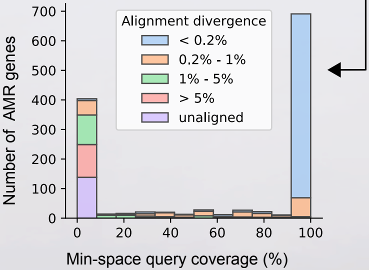
ACATAAATAC ✓

ATAAATACAT ✓

AAATACATAC ✗

Min-space query coverage : 2/3

Retrieval of AMR genes



Part 3 (short)

K and U problems

Known problems

Img: freepic

Can outline research plan.

Unknown problems

Img: freepic

Previously thought impossible.

K and U problems

Known problems

Img: freepic

Can outline research plan.
E.g.:

- mDBG
- BCALM2
- REINDEER

Unknown problems

Img: freepic

Previously thought impossible.
E.g.:

- Minia
- BOSS
- pugz

Conclusion

Future directions

In the dBG area:

- Representations of **multiple samples**: REINDEER, BFT, HowDeSBT, MetaGraph, etc.. (Marchet *et al* review in Genome Res'20)
- Efficient storage of **abundances**: Italiano *et al*; Shibuya & Kucherov, . . .
- Best adaptation to **long reads**: wtdbg2, mdBG, Flye, . . .
- **Disk** compression: SPSS, Simplitigs, . . .
- A standard file format: `github.com/Kmer-File-Format`

And advising team projects:

- Metagenomics strain assembly
- Ancient DNA decontamination
- Structural variants detection
- Sequence transformations

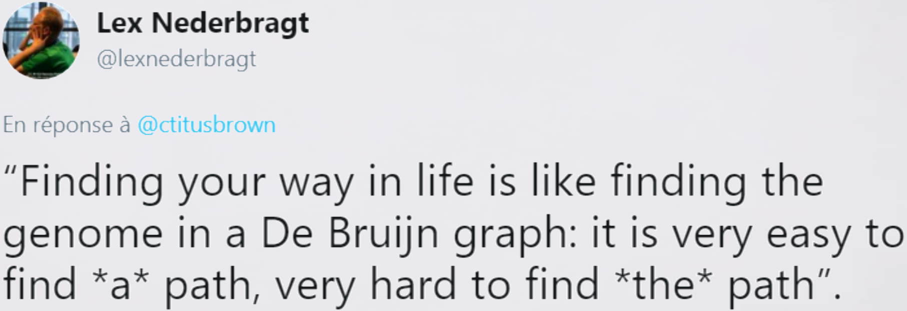
SeqBio Group @ Institut Pasteur

Y. Dufresne, R. Vicedomini, L. Denti, T. Lemane, C. Duitama, L. Blassel

And former students: Camille Marchet, Pierre Marijon, Maël Kerbiriou

And all my current and previous collaborators: I had a wonderful list but it was too long to fit inside this slide <3

Funding: RiSE Pangaia, ITN Alpaca, ANR Inception, ANR Prairie, ANR Transipedia, ANR SeqDigger



Lex Nederbragt
@lexnederbragt

En réponse à @ctitusbrown

“Finding your way in life is like finding the genome in a De Bruijn graph: it is very easy to find *a* path, very hard to find *the* path”.

Thank you all for your attention!

M.	**KUCHEROV Gregory**	Rapporteur	DR,
Mme	**SCHBATH Sophie**	Rapportrice	DR,
M.	**STOYE Jens**	Rapporteur	Professeur,
Mme	**CARBONE Alessandra**	Examinatrice	Professeur,
M.	**GASCUEL Olivier**	Examinateur	DR,
M.	**RICHARD Hugues**	Examinateur	MCF,
M.	**VALLENET David**	Examinateur	DR,
M.	**VINAR Tomas**	Examinateur	Professeur

Bit vector optimality

- A de Bruijn graph **only needs to records the nodes**.
- **Bijection** between **sets of nodes** and **binary vectors** of length 4^k.
- How many different bit vectors of size 4^k and n 1's?

$$\binom{4^k}{n}$$

- Thus, **minimal number of bits** to store a dBG:

$$\log_2(\binom{4^k}{n})$$

Bit vector optimality

- A de Bruijn graph **only needs to records the nodes**.
- **Bijection** between **sets of nodes** and **binary vectors** of length 4^k.
- How many different bit vectors of size 4^k and n 1's?

$$\binom{4^k}{n}$$

- Thus, **minimal number of bits** to store a dBG:

$$\log_2(\binom{4^k}{n})$$

- A compressed bit vector achieves this optimal space.

Bit vector optimality

- A de Bruijn graph **only needs to records the nodes**.
- **Bijection** between **sets of nodes** and **binary vectors** of length 4^k.
- How many different bit vectors of size 4^k and n 1's?

$$\binom{4^k}{n}$$

- Thus, **minimal number of bits** to store a dBG:

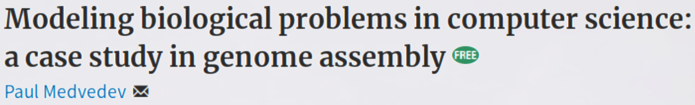
$$\log_2(\binom{4^k}{n})$$

- A compressed bit vector achieves this optimal space.
- (This is much smaller than $O(kn)$, the hash table storage)

Caveats

- Only a subset of approaches were presented
- Ignored query times
- Ignored associated info (e.g. *k*-mer abundances)
- Ignored analysis environment (error-correction, assembly algorithms)
- Ignored multi-*k*
- Ignored reverse-complements
- Ignored the rest of the bioinformatics field, biology, etc..

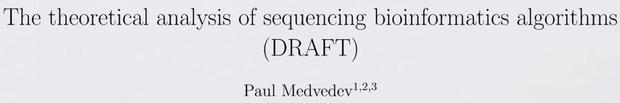
Recommended readings



Modeling biological problems in computer science: a case study in genome assembly FREE

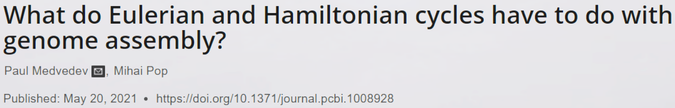
Paul Medvedev ✉

&



The theoretical analysis of sequencing bioinformatics algorithms (DRAFT)

Paul Medvedev[1,2,3]



What do Eulerian and Hamiltonian cycles have to do with genome assembly?

Paul Medvedev ✉, Mihai Pop

Published: May 20, 2021 • https://doi.org/10.1371/journal.pcbi.1008928