kmtricks: Efficient construction of Bloom filters for large sequencing data collections

Téo Lemane, Paul Medvedev, Rayan Chikhi, Pierre Peterlongo

UNIVERSITÉ DE RENNES 1

Inria

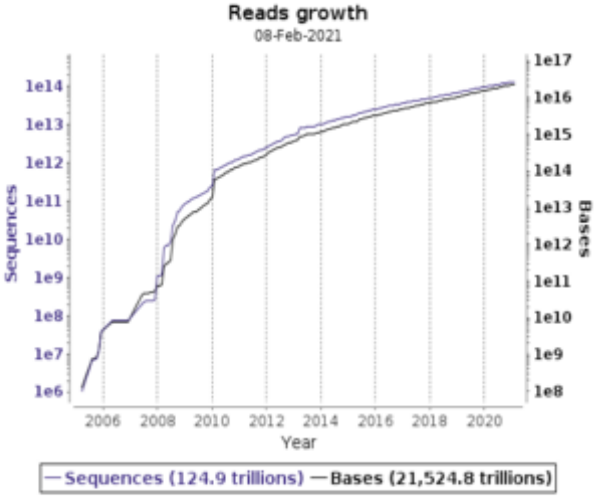
DSB 2021

cnrs

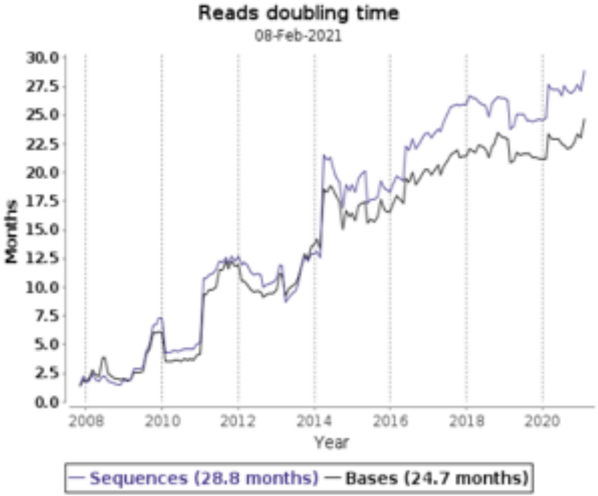
UMR IRISA

Indexing: Motivation

Reads growth



Reads doubling time



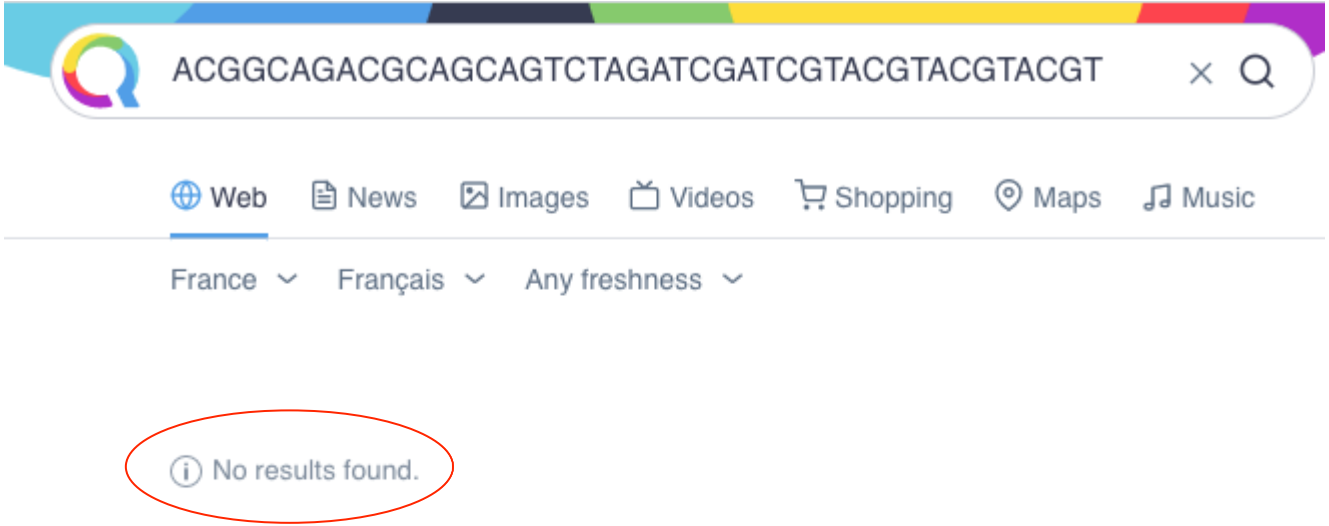
Tara Ocean:
738 billions
paired reads

100000 genome project: **~19 PB**

SRA: **> 50 PB**

https://www.ebi.ac.uk/ena/browser/about/statistics

Indexing: Motivation



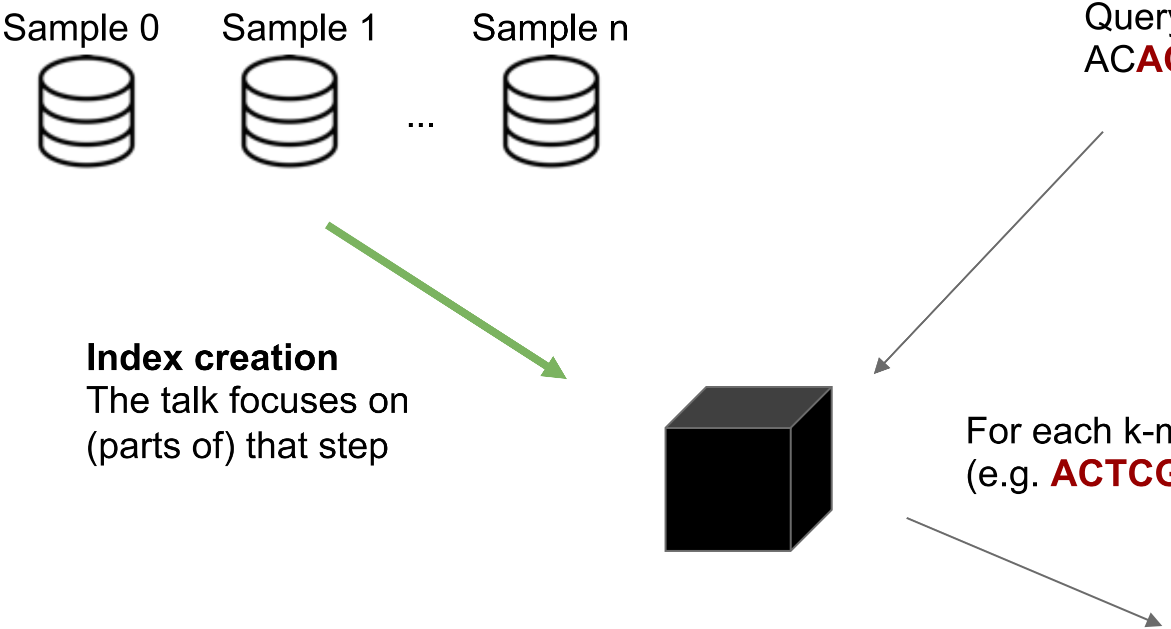
ACGGCAGACGCAGCAGTCTAGATCGATCGTACGTACGTACGT

No results found.

Sequencing data → Assembly/Mapping → Analyses →

Data sleeps in rarely opened drawers

k-mer indexing



Query:
AC**ACTCGCAGAG**GGATTATTTTTAAA

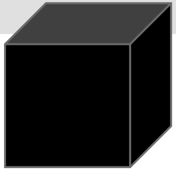
Index creation
The talk focuses on
(parts of) that step

For each k-mer
(e.g. **ACTCGCAGAG**)

sample_0	False
sample_1	True
...	...
sample_n	True

k-mer indexing & bloom filters

Methods:

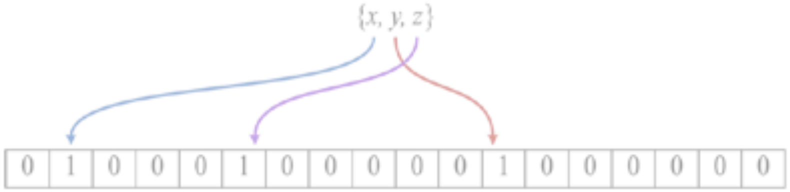


- BFT (Holley *et al.*, 2016)*
- Sequence Bloom Tree*:
  - SBT (Solomon & Kingsford, 2016)
  - AllSomeSBT (Sun *et al.*, 2017)
  - SSBT (Solomon & Kingsford, 2018)
  - HowDeSBT (Harris & Medvedev, 2019)
- Mantis (Pandey *et al.*, 2018)
- SeqOthello (Yu *et al.*, 2018)
- BIGSI (Bradley *et al.*, 2019)*
- COBS (Bingmann *et al.*, 2019)*
- ...

***Based on Bloom filters**

A review: *Data structure based on k-mers for querying large collections of sequencing datasets* (Marchet *et al.*, Genome Research 2020)

Bloom filters



adapted from
wikipedia

It's a bit array $B[0..n]$ with l hash functions

$h_i : \mathcal{U} \to \{0, \ldots, n\} \; \forall i \in [1..l]$

$\text{insert}(x) : B[h_i(x)] \leftarrow 1, \forall i \in [1..l]$

$\text{contains}(x) : \bigwedge_{i=1}^{l} B[h_i(x)]$

Usual Bloom filters construction from k-mers. For each sample:

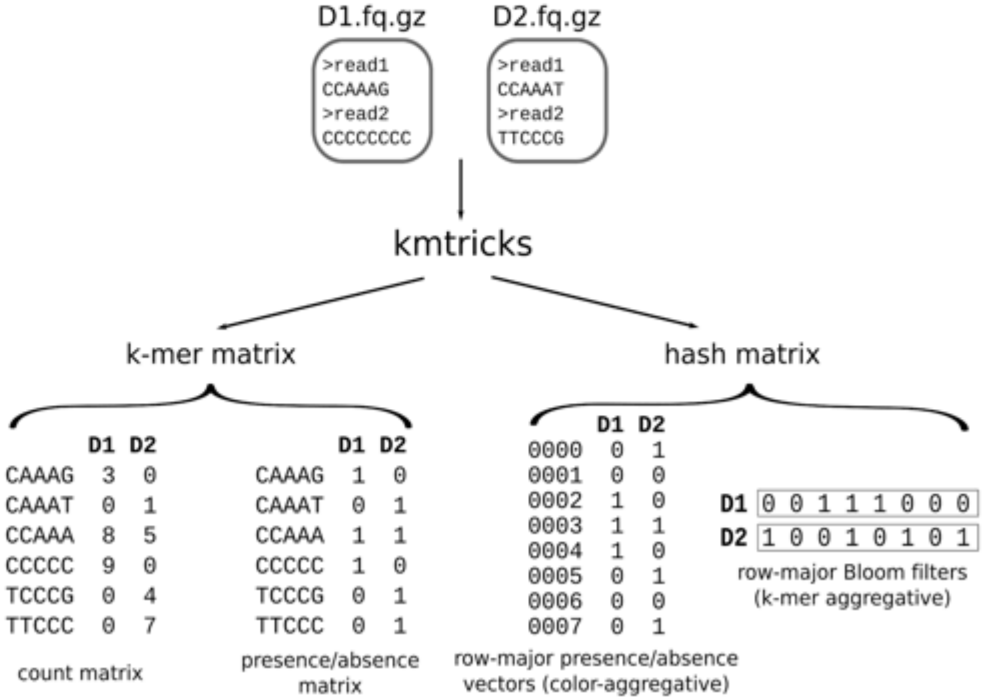
1/2 Count k-mers with efficient k-mer counting tool as Jellyfish (Marçais *et al.*) or KMC (Kokot *et al.*).

2/2 Remove low-abundant k-mers, add others in a Bloom filter

Inefficient + loss of information (low-abundant k-mer removal)
Plan (kmtricks):

- Count hash, not k-mers
- Rescue low-abundant k-mers

kmtricks IOs overview



D1.fq.gz

```
>read1
CCAAAG
>read2
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read2
TTCCCG
```

	D1	D2
CAAAG	3	0
CAAAT	0	1
CCAAA	8	5
CCCCC	9	0
TCCCG	0	4
TTCCC	0	7

count matrix

	D1	D2
CAAAG	1	0
CAAAT	0	1
CCAAA	1	1
CCCCC	1	0
TCCCG	0	1
TTCCC	0	1

presence/absence matrix

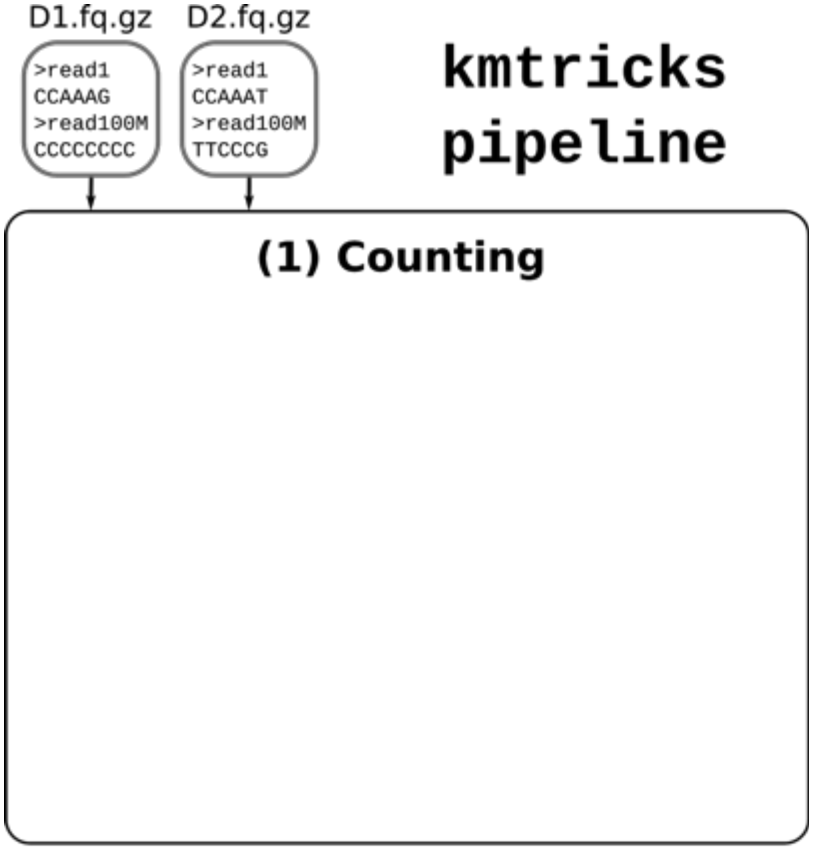
	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

row-major presence/absence vectors (color-aggregative)

D1	0	0	1	1	1	0	0	0
D2	1	0	0	1	0	1	0	1

row-major Bloom filters (k-mer aggregative)

kmtricks pipeline



D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

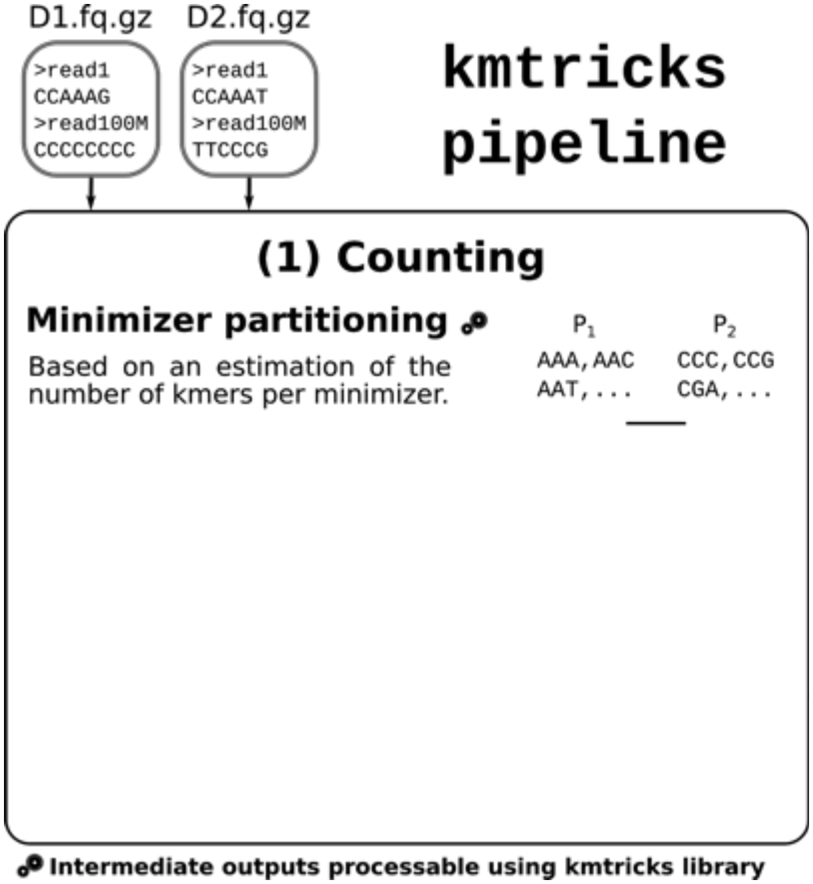
D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

Intermediate outputs processable using kmtricks library

kmtricks pipeline



D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

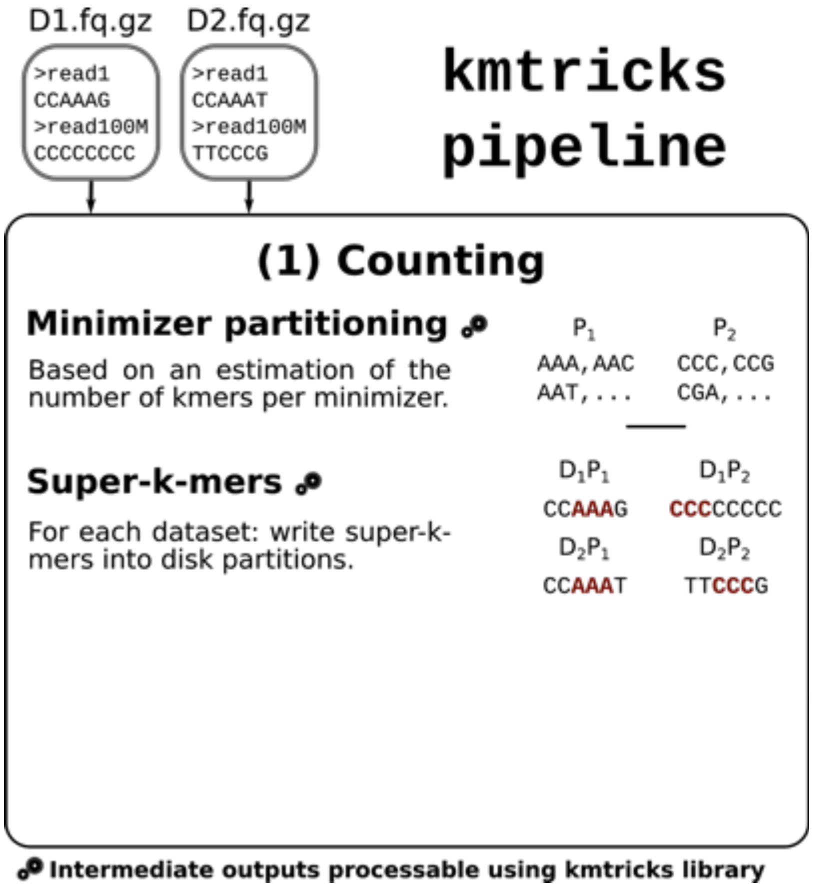
Minimizer partitioning 🔗

Based on an estimation of the number of kmers per minimizer.

P_1	P_2
AAA, AAC	CCC, CCG
AAT, ...	CGA, ...

🔗 **Intermediate outputs processable using kmtricks library**

kmtricks pipeline



D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

P_1	P_2
AAA, AAC AAT, . . .	CCC, CCG CGA, . . .

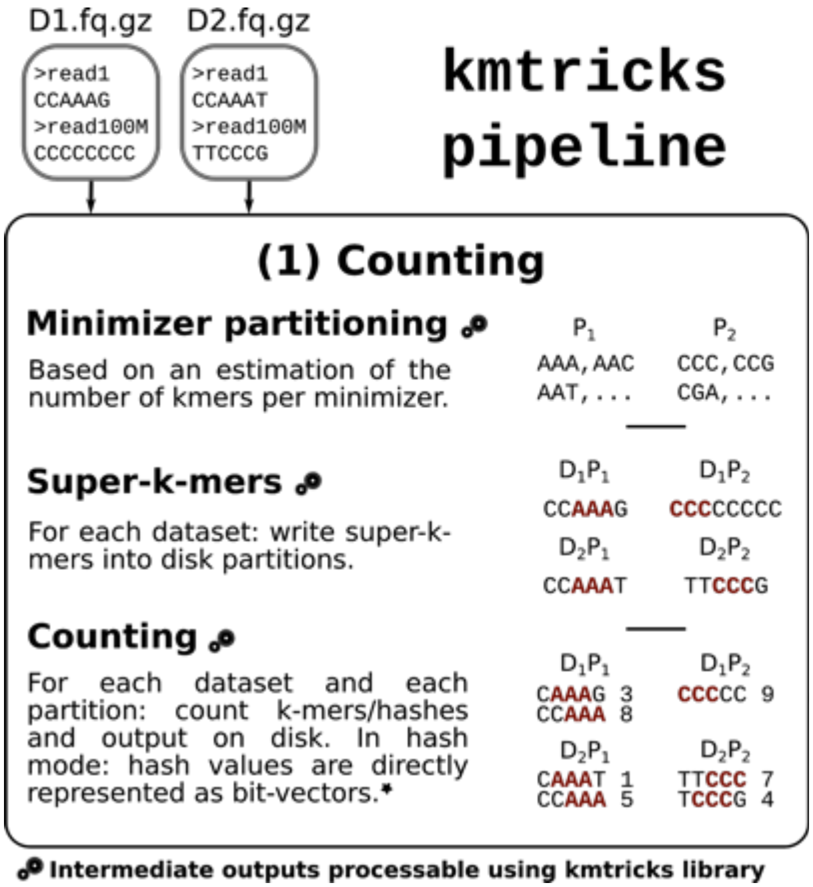
Super-k-mers

For each dataset: write super-k-mers into disk partitions.

D_1P_1	D_1P_2
CC**AAA**G	**CCC**CCCCC
D_2P_1	D_2P_2
CC**AAA**T	TT**CCC**G

Intermediate outputs processable using kmtricks library

kmtricks pipeline



(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

Super-k-mers

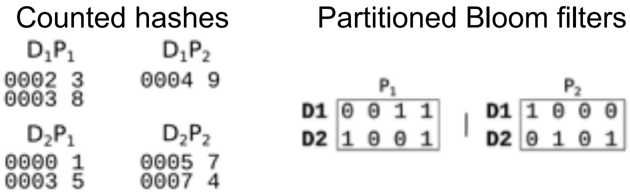
For each dataset: write super-k-mers into disk partitions.

Counting

For each dataset and each partition: count k-mers/hashes and output on disk. In hash mode: hash values are directly represented as bit-vectors.*

Intermediate outputs processable using kmtricks library

kmtricks specific



Counted hashes

D_1P_1
```
0002 3
0003 8
```

D_1P_2
```
0004 9
```

D_2P_1
```
0000 1
0003 5
```

D_2P_2
```
0005 7
0007 4
```

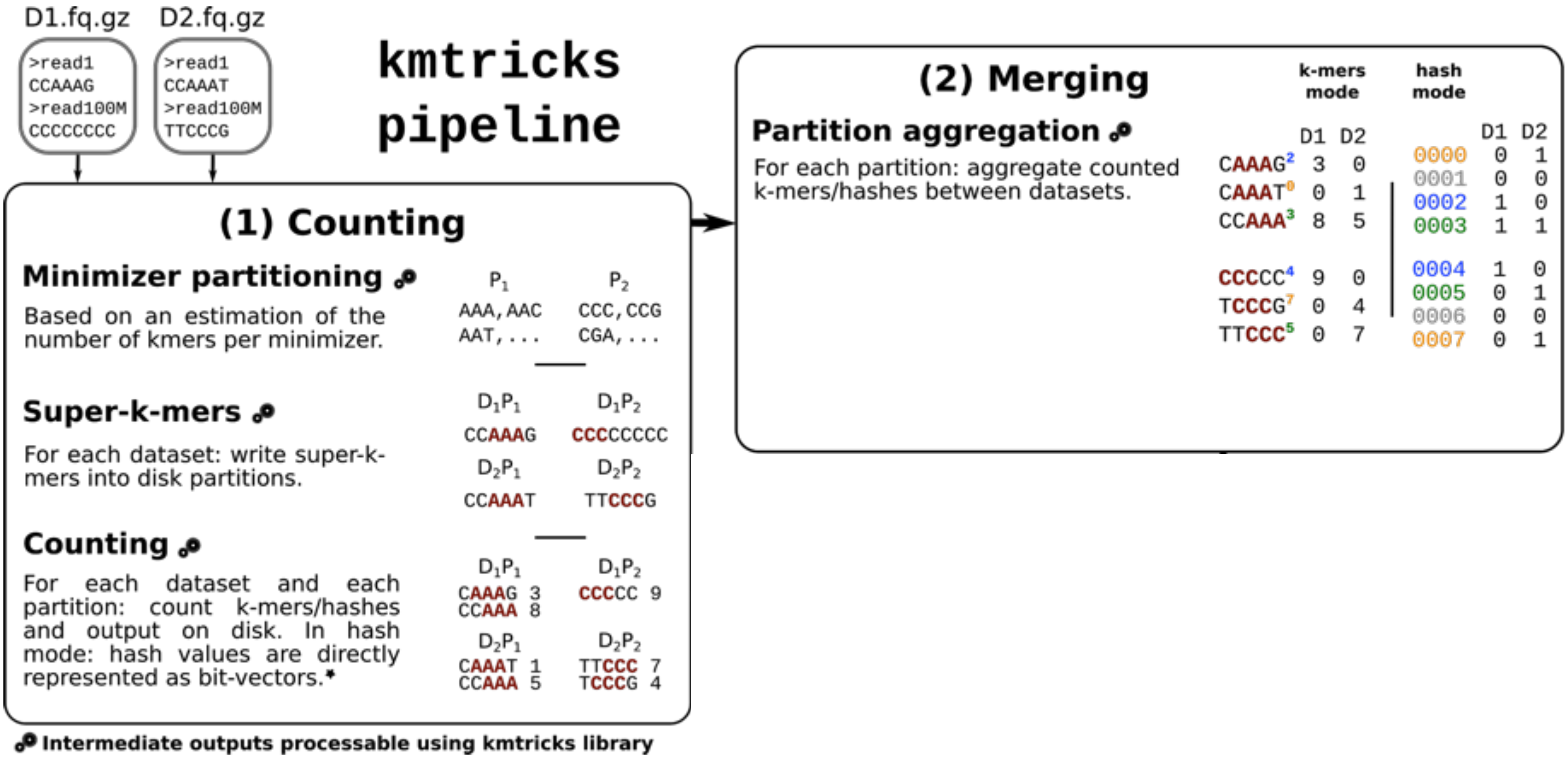
Partitioned Bloom filters

P_1				
D1	0	0	1	1
D2	1	0	0	1

P_2				
D1	1	0	0	0
D2	0	1	0	1

P1: 0-3
P2: 4-7

kmtricks pipeline



D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

P_1	P_2
AAA, AAC AAT, . . .	CCC, CCG CGA, . . .

Super-k-mers

For each dataset: write super-k-mers into disk partitions.

D_1P_1	D_1P_2
CCAAAG	CCCCCCCC
D_2P_1	D_2P_2
CCAAAT	TTCCCG

Counting

For each dataset and each partition: count k-mers/hashes and output on disk. In hash mode: hash values are directly represented as bit-vectors.*

D_1P_1	D_1P_2
CAAAG 3 CCAAA 8	CCCCC 9
D_2P_1	D_2P_2
CAAAT 1 CCAAA 5	TTCCC 7 TCCCG 4

(2) Merging

Partition aggregation

For each partition: aggregate counted k-mers/hashes between datasets.

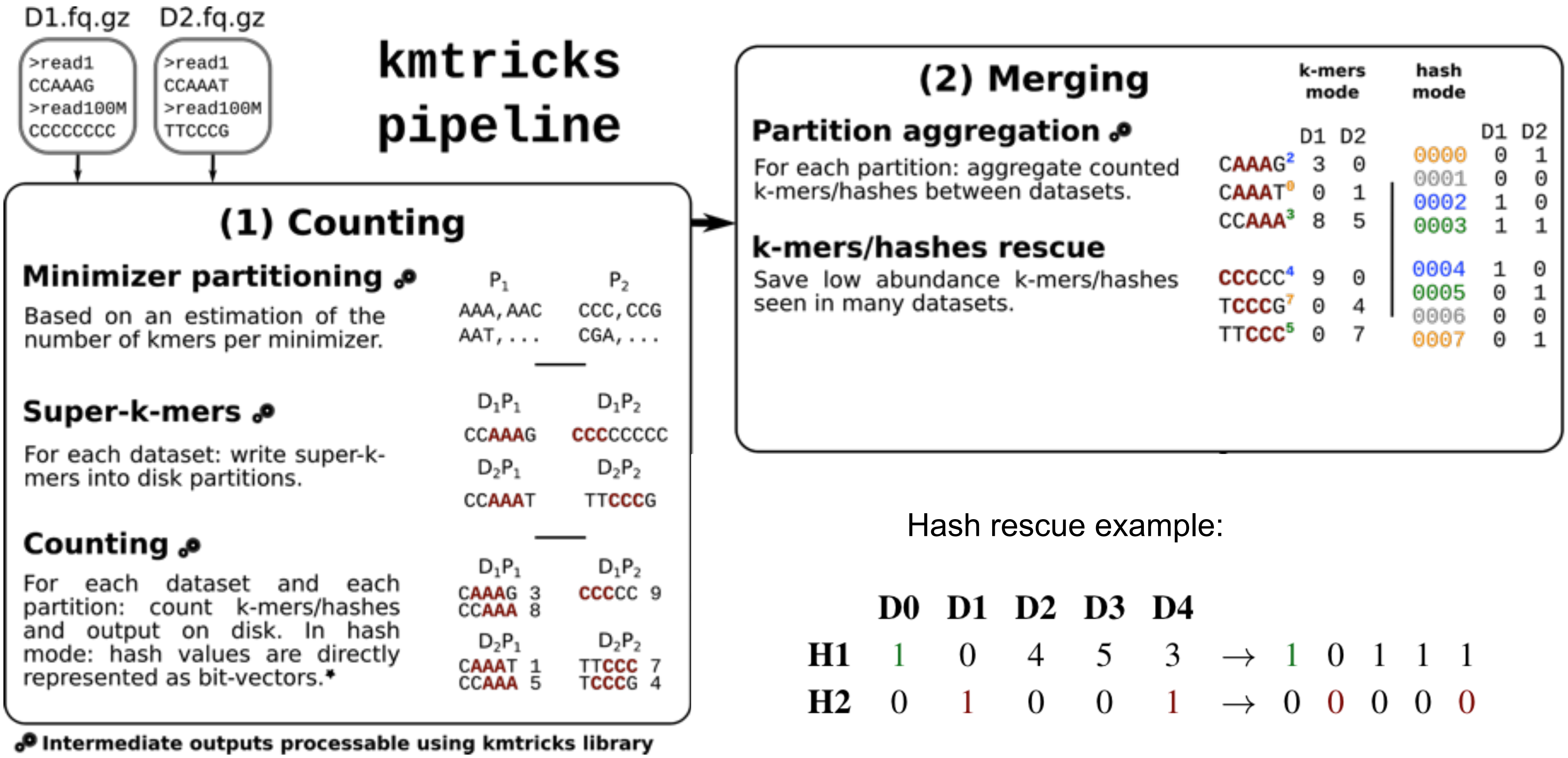
k-mers mode

	D1	D2
CAAAG^2	3	0
CAAAT^0	0	1
CCAAA^3	8	5
CCCCC^4	9	0
TCCCG^7	0	4
TTCCC^5	0	7

hash mode

	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

Intermediate outputs processable using kmtricks library



kmtricks pipeline

(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

Super-k-mers

For each dataset: write super-k-mers into disk partitions.

Counting

For each dataset and each partition: count k-mers/hashes and output on disk. In hash mode: hash values are directly represented as bit-vectors.*

Intermediate outputs processable using kmtricks library

(2) Merging

Partition aggregation

For each partition: aggregate counted k-mers/hashes between datasets.

k-mers/hashes rescue

Save low abundance k-mers/hashes seen in many datasets.

k-mers mode

	D1	D2
CAAAG^{2}	3	0
CAAAT^{0}	0	1
CCAAA^{3}	8	5
CCCCC^{4}	9	0
TCCCG^{7}	0	4
TTCCC^{5}	0	7

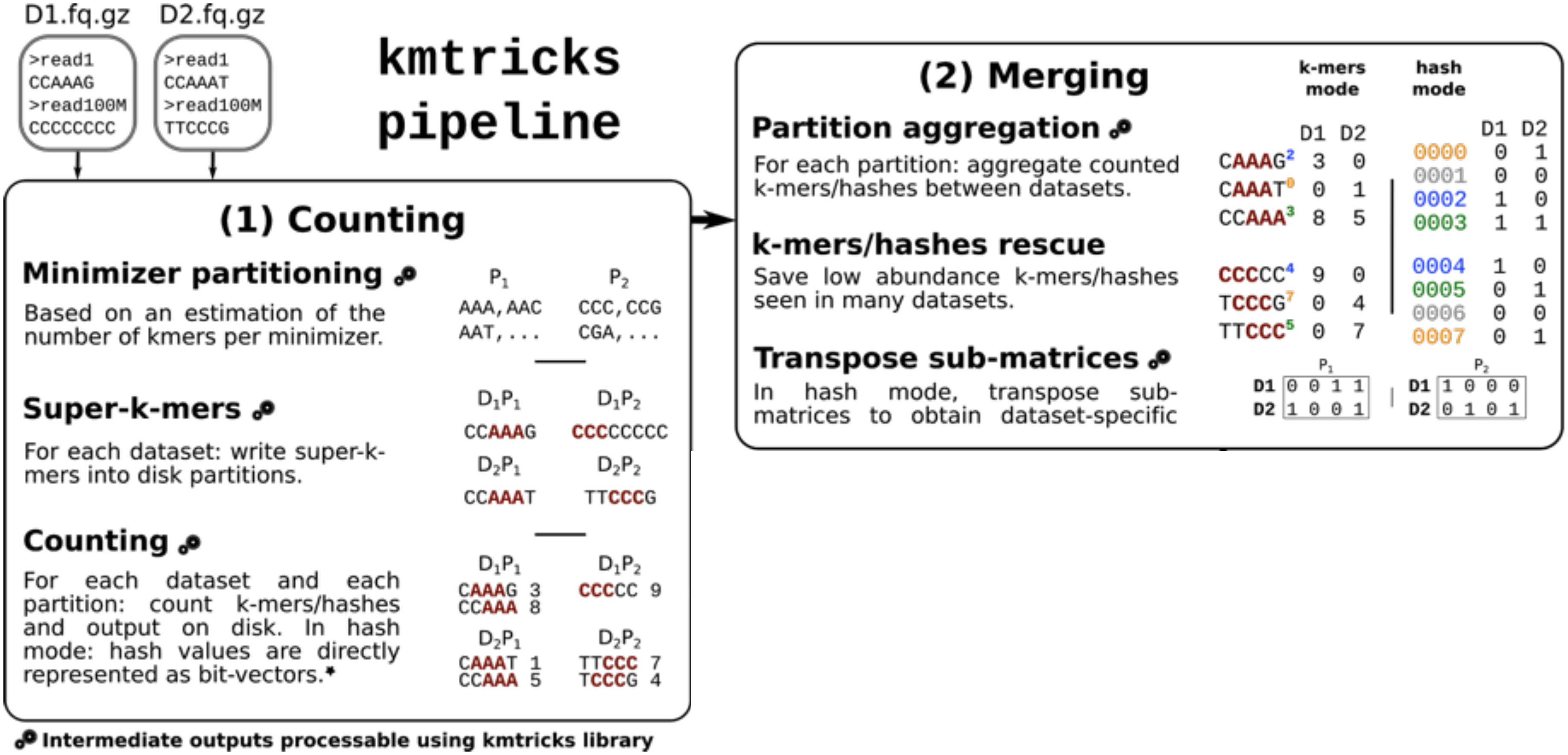
hash mode

	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

Hash rescue example:

	D0	D1	D2	D3	D4						
H1	1	0	4	5	3	→	1	0	1	1	1
H2	0	1	0	0	1	→	0	0	0	0	0

kmtricks pipeline



D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

P$_1$	P$_2$
AAA, AAC AAT, . . .	CCC, CCG CGA, . . .

Super-k-mers

For each dataset: write super-k-mers into disk partitions.

D$_1$P$_1$	D$_1$P$_2$
CCAAAG	CCCCCCCC
D$_2$P$_1$	D$_2$P$_2$
CCAAAT	TTCCCG

Counting

For each dataset and each partition: count k-mers/hashes and output on disk. In hash mode: hash values are directly represented as bit-vectors.*

D$_1$P$_1$	D$_1$P$_2$
CAAAG 3 CCAAA 8	CCCCC 9
D$_2$P$_1$	D$_2$P$_2$
CAAAT 1 CCAAA 5	TTCCC 7 TCCCG 4

(2) Merging

Partition aggregation

For each partition: aggregate counted k-mers/hashes between datasets.

k-mers/hashes rescue

Save low abundance k-mers/hashes seen in many datasets.

Transpose sub-matrices

In hash mode, transpose sub-matrices to obtain dataset-specific

k-mers mode

	D1	D2
CAAAG^2	3	0
CAAAT^0	0	1
CCAAA^3	8	5
CCCCCC^4	9	0
TCCCG^7	0	4
TTCCC^5	0	7

hash mode

	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

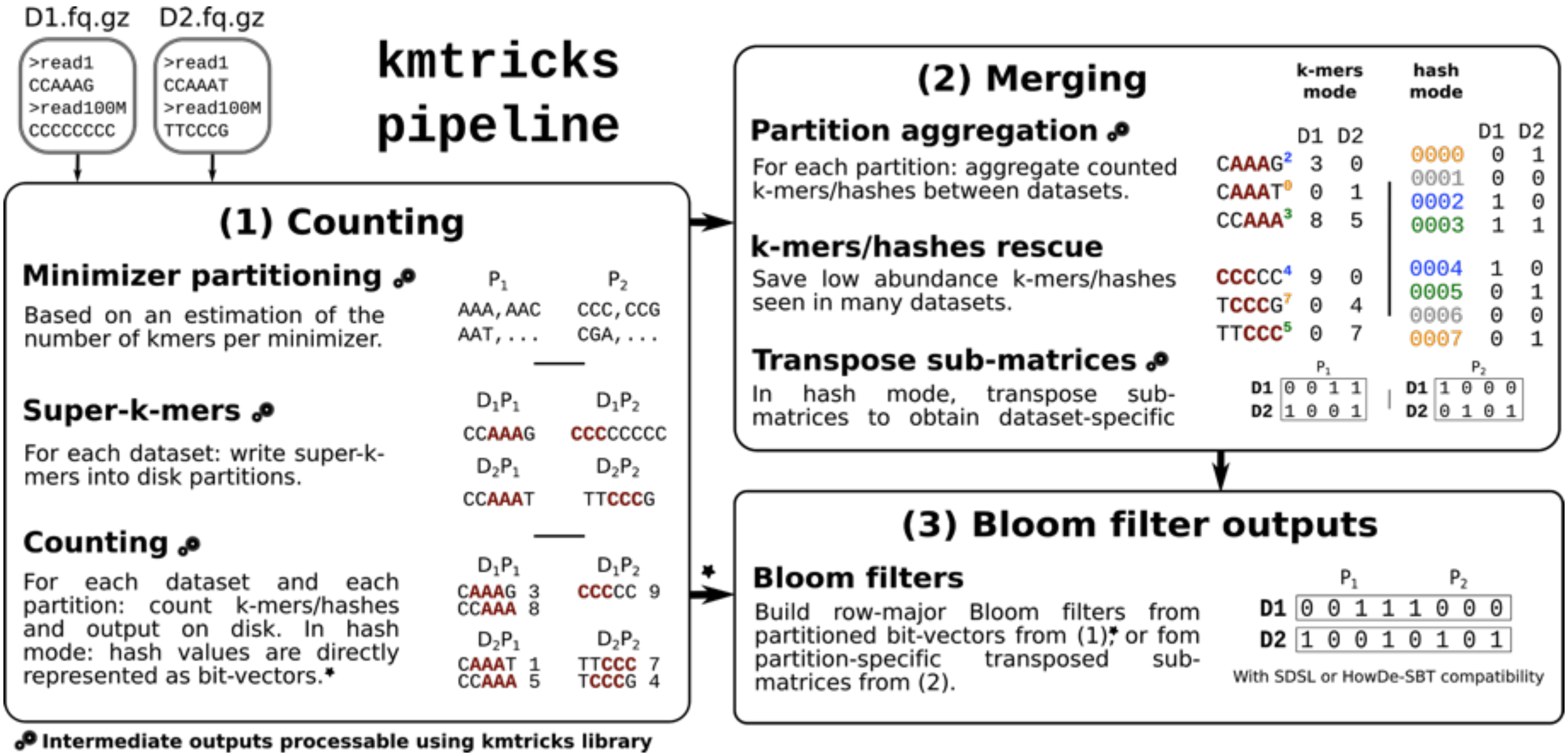
P$_1$

D1	0	0	1	1
D2	1	0	0	1

P$_2$

D1	1	0	0	0
D2	0	1	0	1

Intermediate outputs processable using kmtricks library



kmtricks pipeline

D1.fq.gz

```
>read1
CCAAAG
>read100M
CCCCCCCC
```

D2.fq.gz

```
>read1
CCAAAT
>read100M
TTCCCG
```

(1) Counting

Minimizer partitioning

Based on an estimation of the number of kmers per minimizer.

P_1	P_2
AAA, AAC	CCC, CCG
AAT, . . .	CGA, . . .

Super-k-mers

For each dataset: write super-k-mers into disk partitions.

D_1P_1	D_1P_2
CCAAAG	CCCCCCCC
D_2P_1	**D_2P_2**
CCAAAT	TTCCCG

Counting

For each dataset and each partition: count k-mers/hashes and output on disk. In hash mode: hash values are directly represented as bit-vectors.*

D_1P_1	D_1P_2
CAAAG 3 CCAAA 8	CCCCC 9
D_2P_1	**D_2P_2**
CAAAT 1 CCAAA 5	TTCCC 7 TCCCG 4

(2) Merging

Partition aggregation

For each partition: aggregate counted k-mers/hashes between datasets.

k-mers/hashes rescue

Save low abundance k-mers/hashes seen in many datasets.

Transpose sub-matrices

In hash mode, transpose sub-matrices to obtain dataset-specific

k-mers mode

	D1	D2
CAAAG[2]	3	0
CAAAT[0]	0	1
CCAAA[3]	8	5
CCCCC[4]	9	0
TCCCG[7]	0	4
TTCCC[5]	0	7

hash mode

	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

P_1

D1	0	0	1	1
D2	1	0	0	1

P_2

D1	1	0	0	0
D2	0	1	0	1

(3) Bloom filter outputs

Bloom filters

Build row-major Bloom filters from partitioned bit-vectors from (1),* or fom partition-specific transposed sub-matrices from (2).

	P_1				P_2			
D1	0	0	1	1	1	0	0	0
D2	1	0	0	1	0	1	0	1

With SDSL or HowDe-SBT compatibility

Intermediate outputs processable using kmtricks library

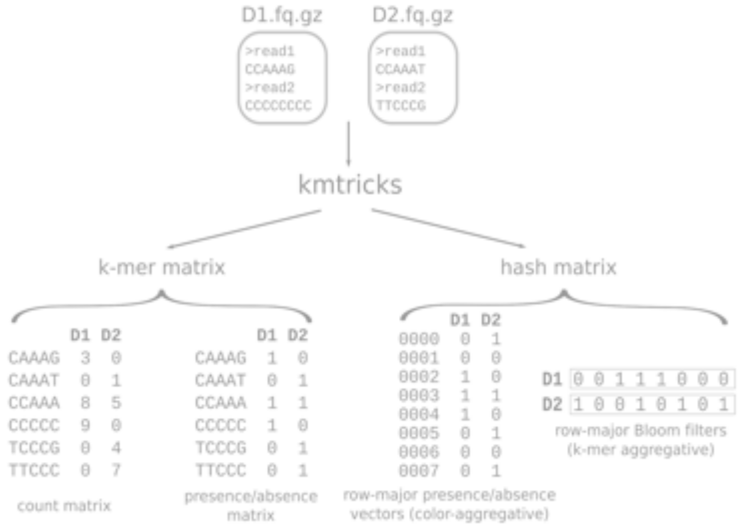
Applications

- kmtricks (standalone)
  - **k-mer matrix** construction
  - **Bloom filter** construction
  - With / wo rescue

Up next:

kmtricks + HowDeSBT

  - kmtricks + compatible HowDeSBT software available on kmtrick’s Github



```
>read1
CCAAAG
>read2
CCCCCCCC
```

```
>read1
CCAAAT
>read2
TTCCCG
```

	D1	D2
CAAAG	3	0
CAAAT	0	1
CCAAA	8	5
CCCCC	9	0
TCCCG	0	4
TTCCC	0	7

count matrix

	D1	D2
CAAAG	1	0
CAAAT	0	1
CCAAA	1	1
CCCCC	1	0
TCCCG	0	1
TTCCC	0	1

presence/absence matrix

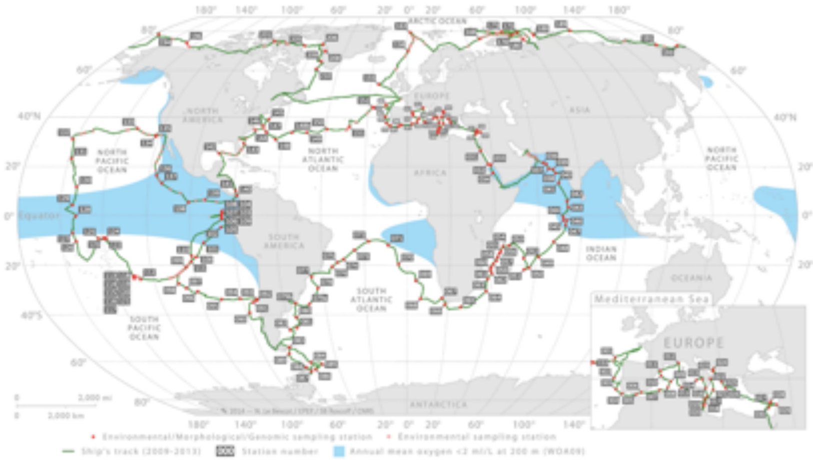
	D1	D2
0000	0	1
0001	0	0
0002	1	0
0003	1	1
0004	1	0
0005	0	1
0006	0	0
0007	0	1

row-major presence/absence vectors (color-aggregative)

D1	0	0	1	1	1	0	0	0
D2	1	0	0	1	0	1	0	1

row-major Bloom filters (k-mer aggregative)

kmtricks performances on Tara Ocean

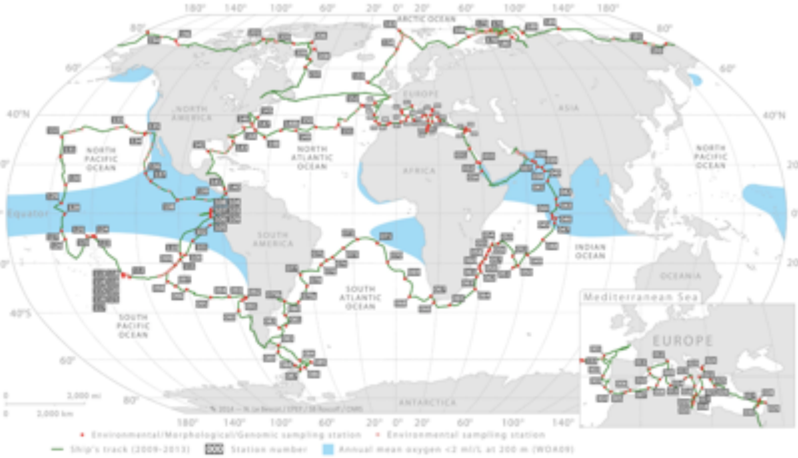


Pesant *et al.*

241 bacterial metagenomic stations:

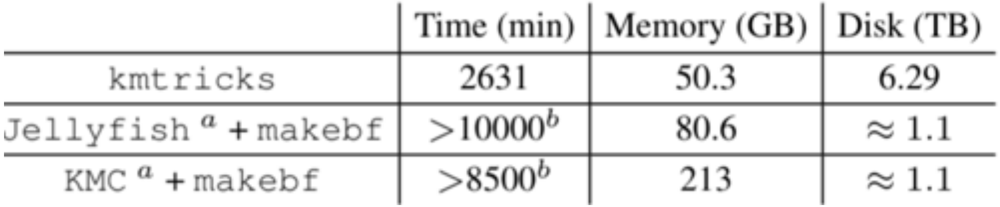
- 712 fastq.gz
- > 6.5 TB
- 266 billions distinct k-mers
- 174 millions occur twice or more

kmtricks performances on Tara Ocean



Pesant *et al.*

Performances for bloom filter creation:



	Time (min)	Memory (GB)	Disk (TB)
kmtricks	2631	50.3	6.29
Jellyfish[a] + makebf	>10000[b]	80.6	≈ 1.1
KMC[a] + makebf	>8500[b]	213	≈ 1.1

kmtricks: hash rescue step - first conserving all kmers

241 bacterial metagenomic stations:

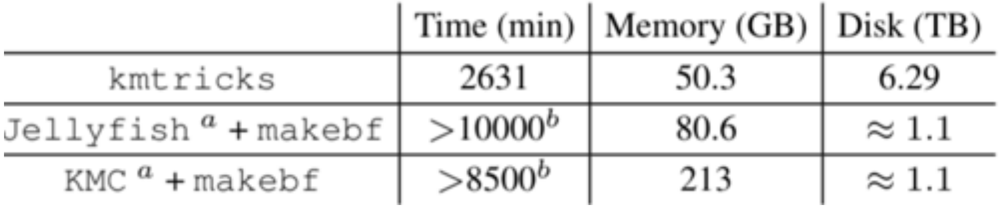
- 712 fastq.gz
- > 6.5 TB
- 266 billions distinct k-mers
- 174 millions occur twice or more

kmtricks + HowDeSBT performances on Tara Ocean

ARCTIC OCEAN
NORTH AMERICA
NORTH PACIFIC OCEAN
NORTH ATLANTIC OCEAN
EUROPE
ASIA
AFRICA
SOUTH AMERICA
SOUTH PACIFIC OCEAN
SOUTH ATLANTIC OCEAN
INDIAN OCEAN
OCEANIA
ANTARCTICA
Mediterranean Sea
Equator
0 2,000 mi
0 2,000 km
Environmental/Morphological/Genomic sampling station
Environmental sampling station
Ship's track (2009-2013)
Station number
Annual mean oxygen <2 ml/L at 200 m (WOA09)

Pesant *et al.*

Performances for bloom filter creation:



	Time (min)	Memory (GB)	Disk (TB)
kmtricks	2631	50.3	6.29
Jellyfish [a] + makebf	>10000[b]	80.6	≈ 1.1
KMC [a] + makebf	>8500[b]	213	≈ 1.1

kmtricks: hash rescue step - first conserving all kmers

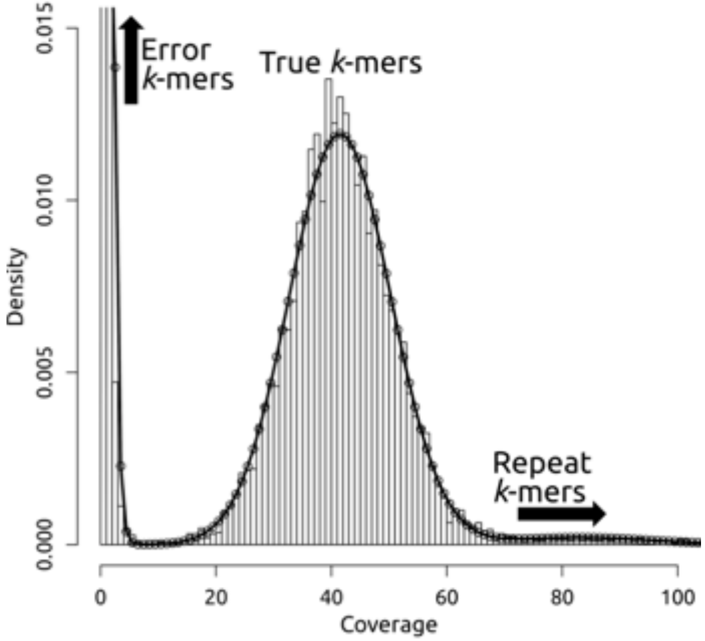
241 bacterial metagenomic stations:

- 712 fastq.gz
- > 6.5 TB
- 266 billions distinct k-mers
- 174 millions occur twice or more

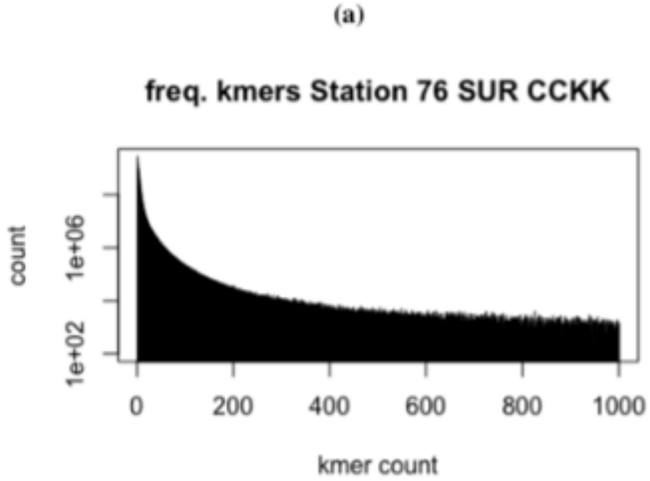
Used with HowDeSBT

HowDeSBT tree construction from BFs: ~40h
Final index size: 612 GB
Query time: 19 minutes (for 1 or 1000 queries)

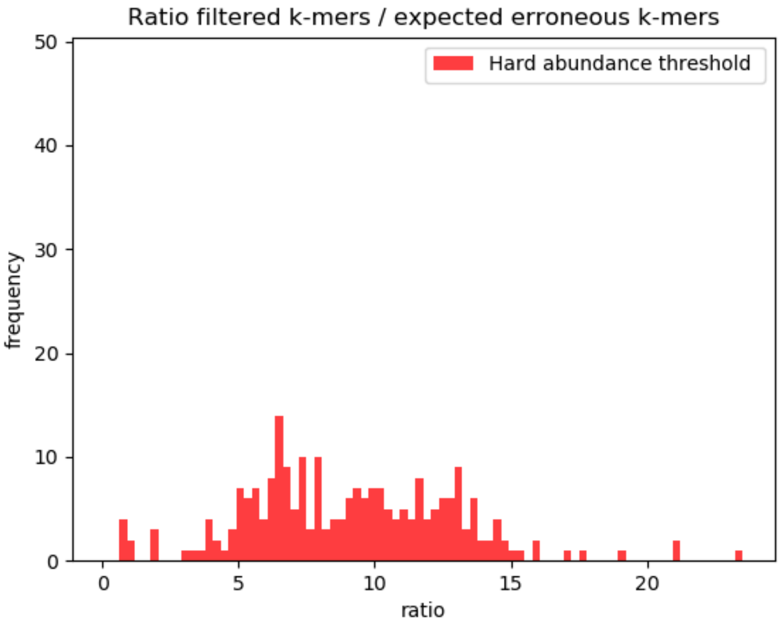
Collection-aware k-mer filtering recovers large amounts of weak signal



Laehnemann *et al.*



Collection-aware k-mer filtering recovers large amounts of weak signal



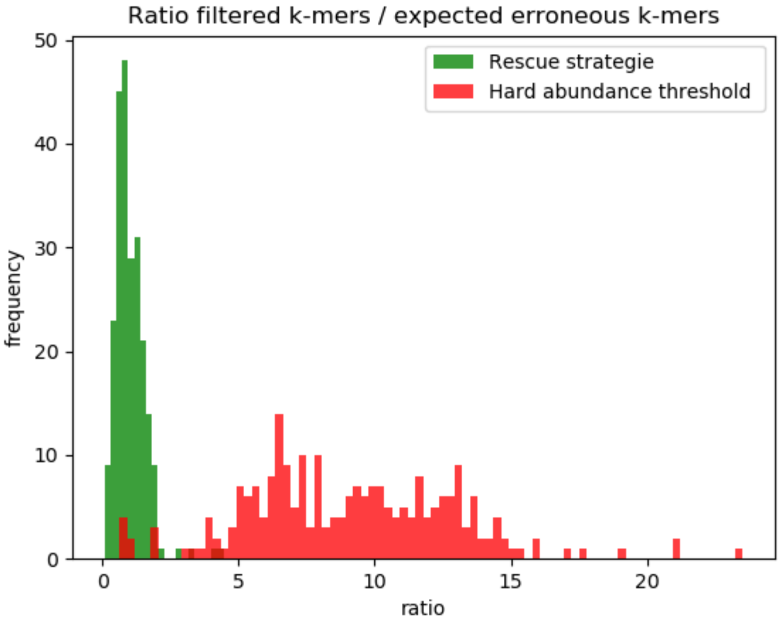
$$\frac{\text{k-mers occuring once}}{\text{Expected number of erroneous k-mers}}$$

1 : 🙂

> 1 : too many k-mers are discarded

< 1 : too many k-mers are kept

Collection-aware k-mer filtering recovers large amounts of weak signal



k-mers occuring once

Expected number of erroneous k-mers

Unrescued k-mers

Expected number of erroneous k-mers

1 : 🙂
> 1 : too many k-mers are discarded
< 1 : too many k-mers are kept

Conclusion & Future work

- Improves bf construction time
  - **still very insufficient to hope to scale up on the very large databases**

- kmtricks + HowDe-SBT -> build an index for medium scale dataset:
  - all bacterial metagenomics reads from Tara Ocean in a few days.

- Low-abundant k-mer rescue procedure seems to recover an interesting signal
  - Future steps: analyze the signal present in recovered k-mers

- Take advantage of partitioned Bloom filters in indexes:
  - Construction time: Parallel construction of HowDe-SBT tree
  - Query time: query one tree per partition

https://github.com/tlemane/kmtricks

Thank you !

https://github.com/tlemane/kmtricks

References

N .Luhmann, et al. Blastfrost: Fast querying of 100,000 s of bacterial genomes in bifrost graphs. BioRxiv, 2020.

R. Wittler. Alignment-and reference-free phylogenomics with colored de bruijn graphs. Algorithms for Molecular Biology, 2020.

G. Holley, R. Wittler, and J. Stoye, “Bloom Filter Trie: An alignment-free and reference-free data structure for pan-genome storage,” *Algorithms Mol. Biol.*, vol. 11, no. 1, p. 3, 2016, doi: 10.1186/s13015-016-0066-8.

B. Solomon and C. Kingsford, “Fast search of thousands of short-read sequencing experiments,” *Nat. Biotechnol.*, vol. 34, no. 3, pp. 300–302, Mar. 2016, doi: 10.1038/nbt.3442.

B. Solomon and C. Kingsford, “Improved search of large transcriptomic sequencing databases using split sequence bloom trees,” in *Journal of Computational Biology*, 2018, vol. 25, no. 7, pp. 755–765, doi: 10.1089/cmb.2017.0265.

C. Sun, R. S. Harris, R. Chikhi, and P. Medvedev, “AllSome Sequence Bloom Trees,” *J. Comput. Biol.*, vol. 25, no. 5, pp. 467–479, 2018, doi: 10.1089/cmb.2017.0258.

R. S. Harris and P. Medvedev, “Improved representation of sequence Bloom trees,” *Bioinformatics*, 2019, doi: 10.1093/bioinformatics/btz662.

References

P. Pandey, F. Almodaresi, M. A. Bender, M. Ferdman, R. Johnson, and R. Patro, “Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index,” *Cell Syst.*, vol. 7, no. 2, pp. 201-207.e4, Aug. 2018, doi: 10.1016/j.cels.2018.05.021.

Y. Yu *et al.*, “SeqOthello: querying RNA-seq experiments at scale,” *Genome Biol.*, vol. 19, no. 1, p. 167, Oct. 2018, doi: 10.1186/s13059-018-1535-9.

P. Bradley, H. C. den Bakker, E. P. C. Rocha, G. McVean, and Z. Iqbal, “Ultrafast search of all deposited bacterial and viral genomic data,” *Nat. Biotechnol.*, vol. 37, no. 2, pp. 152–159, Feb. 2019, doi: 10.1038/s41587-018-0010-1.

T. Bingmann, P. Bradley, F. Gauger, and Z. Iqbal, “COBS: a Compact Bit-Sliced Signature Index,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11811 LNCS, pp. 285–303, May 2019.

C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. Chikhi, “Data structures based on *k* -mers for querying large collections of sequencing data sets,” *Genome Res.*, vol. 31, no. 1, pp. 1–12, Jan. 2021, doi: 10.1101/gr.260604.119.