Data-structures for querying large k-mer (collections of) sets

Séminaire de la KIM Data & Life Sciences
14 février 2022

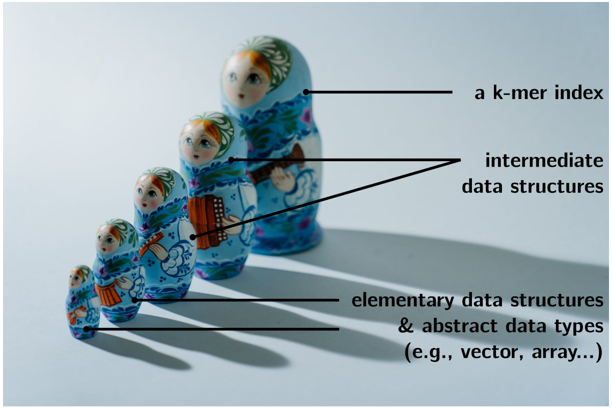
Camille Marchet
CNRS, CRIStAL Lille, France

camille.marchet@univ-lille.fr

@CamilleMrcht

Introduction - Data structures

- Russian dolls
  - Legos or "building blocks"
  - *Abstract data types*: set, multi-set, list...
- Existence (or not) of an implementation


a k-mer index
intermediate data structures
elementary data structures & abstract data types (e.g., vector, array...)

adapted from copyright free, @cottonbro on Pexel

Introduction - Data structures

Data structures are purposeless without **operations**

- Test for emptiness
- Add/delete elements
- Check membership of an element
- Go over all elements

Operations go hand in hand with notions of cost and complexity

- Computation time
- Size in memory

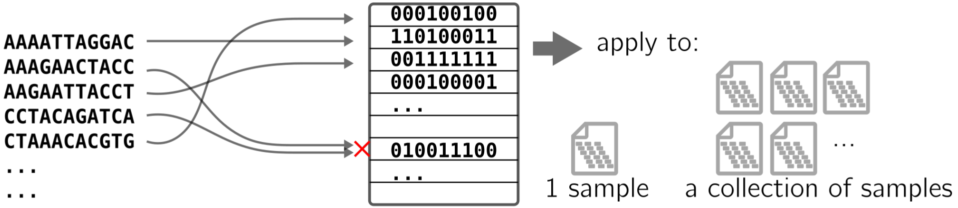
Introduction - When it comes to *k*-mers

	human	*Pinus taeda*	*Ambystoma mexicanum*	*Paris japonica*	metagenomics
# 31-mers	3.2G	10.5G	18.5G	~150G	...

In practice :

- *k* sizes : 11-15 (long reads), 21-51 (short reads)
- Billions (of distinct *k*-mers) easily reached in experiments
- The notion of cost becomes central

Introduction - Themes of this talk



key set (*k*-mers) efficient **representation**

associativity × deal with collisions

values compression

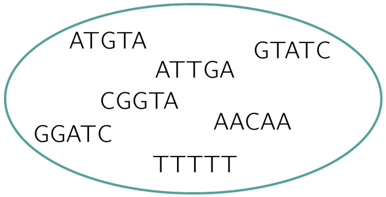
Introduction - *k*-mer data structures

These data structures are foundations of many applications:

- alignment methods
- alignement-free methods (pseudo-alignement, quantification, taxonomic assignation, ecological distances. . .)
- quality analysis, read correction
- representation and usage of graphs (assembly, variant calling, . . .)

Data structures for *k*-mers sets

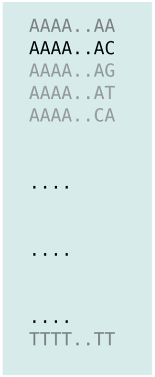
Representation of *k*-mer sets



k-**mer sets** - a naive encoding

4^k possible *k*-mers
~4.10^{18} 31-mers

A	00
C	01
G	10
T	11



using integers

2 bits x *k*
62 bits per 31-mers
→ use 64 bits integers

... but only 3.10^9 31-mers in the human genome

24GB for distinct 31-mers of the whole human genome

k-mer sets - Conway & Bromage[1]

n = ~3.10^9 31-mers
in human genome
uniformely distributed

```
AAAA..AC
ACTT..AG
AGGG..AT
CCTA..CA

....
....
TAGT..TT
```

using
Conway &
Bromage

compressed array

```
0000000
0000001
0000010
0000011
0000101
```

$\binom{4^k}{n}$ bits = 35 bits per 31-mer

~**13 GB** for distinct 31-mers of the whole genome

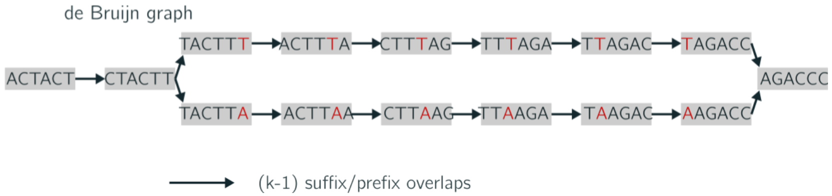
worst case lower bound

[1] Conway & Bromage 2011

k-**mer sets** - Representation using a de Bruijn graph[2]

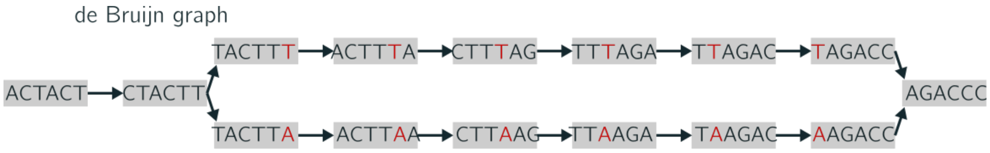
k-mer set

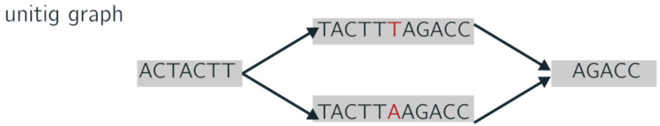
ACTACT TACTTA
CTACTT ACTTAA
TACTTT CTTAAG
ACTTTA TTAAGA
CTTTAG TAAGAC
TTTAGA AAGACC
TTAGAC TAGACC
AGACCC



[2] I use the *node-centric* definition of a DBG

k-mer sets - Representation using unitigs





- Compacted de Bruijn graph

k-**mer sets** - Representation using unitigs

k-mer set | unitig set

```
ACTACT
 CTACTT
  TACTTA
   ACTTAC
    CTTACA
     TTACAG
```

?

k-**mer sets** - Representation using unitigs

k-mer set

```
ACTACT
 CTACTT
  TACTTA
   ACTTAC
    CTTACA
     TTACAG
```

unitig set

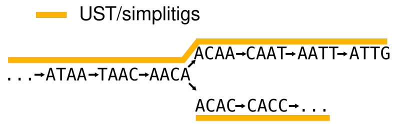
ACTACTTACAG

6 x 6 = 36 bases

6 + 5 = 11 bases

k-**mer sets** - Spectrum preserving string sets

- Other sequences extracted from the DBG are now used
- Spectrum preserving string sets (SPSS)[3]



{ATAACAATTG, ACACC} 15 nucleotides
(other possibility: {ATAACACC, ACAATTG} 15 nucleotides)

- [Rahman et al. 2020, Brinda et al. 2020] Greedy algorithm, nearly optimal
- Applications: de Bruijn graph implementation, alignment
- Open question: constraints on SPSS

[3]A brief description of several SPSS: `https://kamimrcht.github.io/webpage/tigs.html`

k-**mer sets** - encoding with SPSS[4]

n = ~3.10^9 31-mers
in human genome
redundance

```
AAAA..AC
 AAA..ACC
  AA..ACCT

CCTA..CA
 CTA..CAG
....
....
```

using
SPSS

```
AAAA...ACCT

CCTA..CAG
```

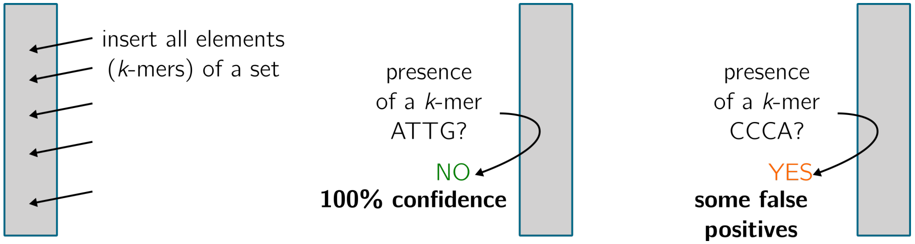
4.1 bits per *k*-mer on average

~**1.5 GB** for distinct 31-mers of the whole human genome

[4]Recently better results by indexing *k*-mer multisets: matchtigs [Schmidt et al. 2021]

k-**mers sets** - Probabilistic representation

Bloom filters [Bloom 1970]

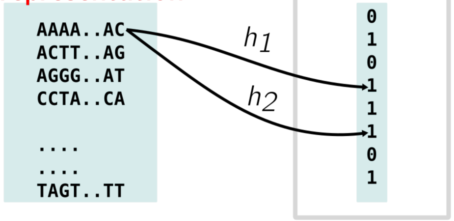

insert all elements (*k*-mers) of a set
presence of a *k*-mer ATTG?
NO
100% confidence
presence of a *k*-mer CCCA?
YES
some false positives

- Extreme simplicity in terms of implementation: a bit array + hash functions
- Very quick to build & query
- Applications: assembly (Minia, Abyss, Hifi-asm), *k*-mer counting (Jellyfish), alignment (Minimap)

k-**mer sets** - encoding with Bloom filters

n = ~3.10^9 31-mers
in human genome
approximate
representation

using
Bloom filters



for a 10 x n BF size

1bit x h x 3.10^{10} bits
= **3.8 GB** for distinct 31-mers
for h=1, **FPR ~10%**

= **11.2 GB** if **0.1% FPR** (h=3)
+ possible further compression

k-**mer sets** - Full text methods?

BWT-based methods rely on the **lexicographic context** of nucleotides to provide a compressed representation

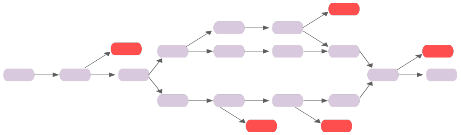
- Need a text such as a genome as an input
- More expressive query, but comes with "unuseful" information for a set

k-**mer sets**- Summary

- SPSS-based methods rely on the **genomic context** ("assemblability") of nucleotides to provide a compacted representation
  - Will structure a *k*-mer set according to the underlying genome
- Bloom filters remain a very popular option

de Bruijn graphs:

- Can be seen as objects to:
  1. assemble sequences
  2. represent a *k*-mer set and to structure the redundancy of datasets in some way
- Interesting feature: facilitate error correction/filtering → impact on performances



Data structures for k-mers sets

Associative indexes for k-mer sets

ATGTA	6
GTATC	127
ATTGA	2
CGGTA	53
AACAA	55
TTTTT	272

k-mers sets index - full-text methods

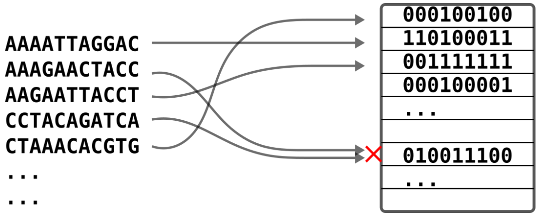
Full-text methods based on the BTW, with the same limitations as previously stated:

- FM-index [Ferragina & Manzini 2004], r-index [Gagie et al. 2017] (improves on space complexity)

Use the paths of the de Bruijn graph as a text, then index the *k*-mers:

- BOSS [Bowe et al. 2012]: a FM-index specialized for *k*-mers
- Applications: indexing large collections of bacterial datasets [Muggli et al. 2017,2019], implementing de Bruijn graphs [Boucher et al. 2015; Karasikov et al. 2021]
- Main downsides:
  - Hypothesis on the paths lengths
  - Slower query in comparison to other approaches

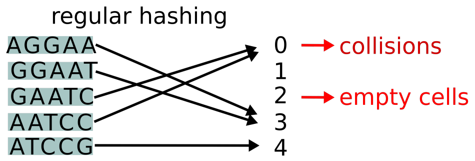
k-**mers sets index** - collisions: a big issue

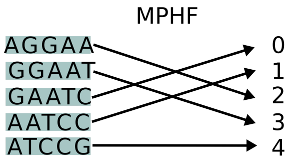


Solutions:

- Pointers: decrease space performances
- Open addressing: decreases time performances (loss of locality)
- A third way...

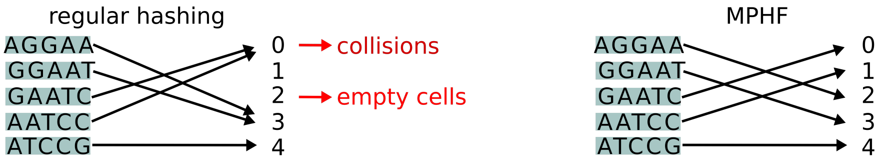
k-mer sets index - Minimal perfect hash functions (MPHF)





- The hash function itself has a cost. Theoretical bound: 1.44 bits per key
- Practical implementations used in bioinformatics (fast construction/query, around 4 bits/key)
  - BBHASH [Limasset et al. 2014]
  - PTHash [Pibiri et al. 2021]

k-**mer sets index** - Minimal perfect hash functions (MPHF)



- ⚠MPHFs are static
- ⚠MPHFs are **only** hash **functions**, in order to build a hash table we need a representation of the keys to deal with alien keys

k-mer sets index - Specialized hash tables

Efficient *k*-mer hash tables:

- Pufferfish: MPHF+unitigs [Almodaresi et al. 2018]
- BLight: MPHF+partitioning+SPSS [Marchet et al. 2019][5]
- Counting quotient filters [Pandey et al. 2017]: another hashing strategy

Applications:

- Example of achievement:
  index the 31-mers of the human genome in RAM in <8GB (BLight)
- Counting *k*-mers [Pandey et al. 2018]
- Large scale quantification [Marchet et al. 2020]
- Read alignment [Almodaresi et al. 2021]

[5] and recently SSHash, [Pibiri 2022]

Summary on indexing k-mers sets

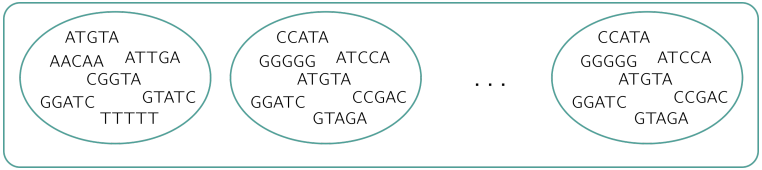
- Full-text (BWT-based) or hashing+SPSS appear to be the two major ways for indexing k-mers
  - Tradeoff: BWT-based has more expressivity (order preserving) but lower performances in practice (notably query)
- Indexing de Bruijn graphs (+navigational operations, dynamicity) is a field on its own [6]

A survey about all these data structures: Chikhi R, Holub J, Medvedev P. 2019. Data structures to represent a set of k-long DNA sequences. *ACM Computing Surveys*

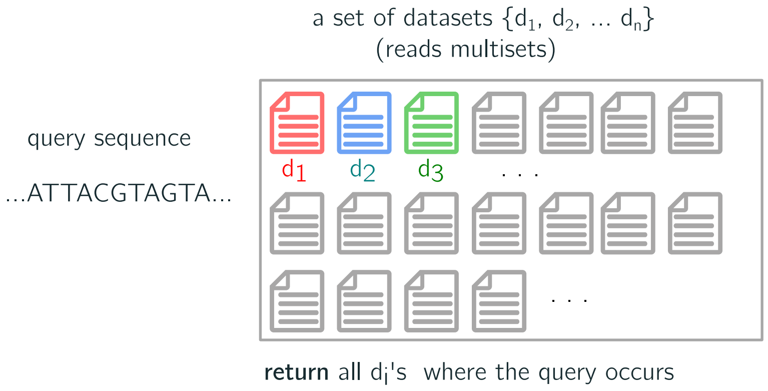
[6] see for instance `http://rayan.chikhi.name/pdf/2021-july-9-cie.pdf`

Data structures for *k*-mers sets

Collections of *k*-mer sets



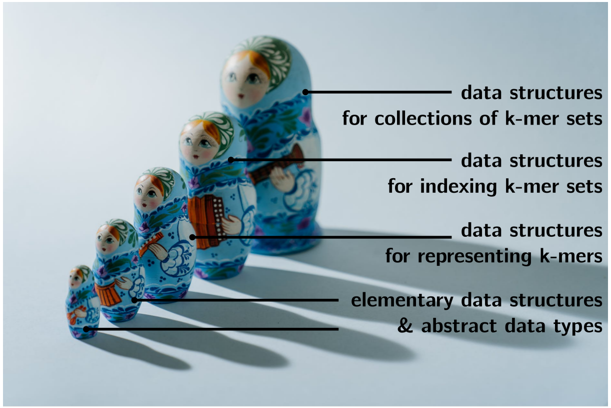
Collections of *k*-mer sets


a set of datasets $\{d_1, d_2, \ldots d_n\}$
(reads multisets)
query sequence
...ATTACGTAGTA...
d_1
d_2
d_3
. . .
. . .
return all d_i's where the query occurs

- Each dataset (and the query) are seen as sets of *k*-mers
- The query is "present" in a dataset if *enough* of its *k*-mers are found

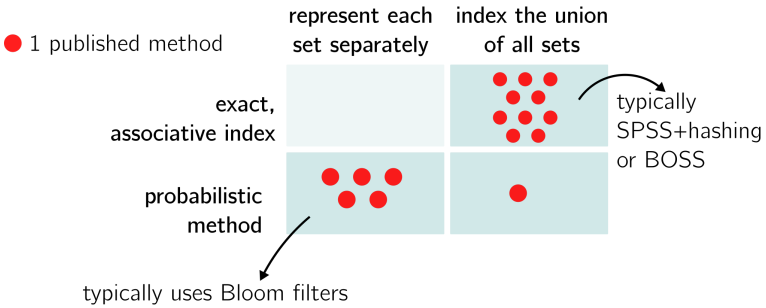
Collections of *k*-mer sets

We need to handle **multiple** sets of *k*-mers and query the presence/absence of a sequence



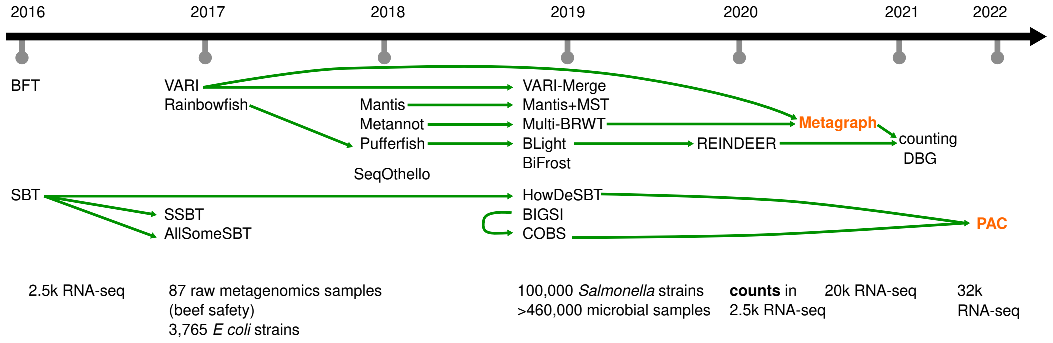
adapted from copyright free, @cottonbro on Pexel

Collections of *k*-mer sets - State of the art


● 1 published method
represent each set separately
index the union of all sets
exact, associative index
probabilistic method
typically SPSS+hashing or BOSS
typically uses Bloom filters

- Exact methods: for precise, short queries or when colored de Bruijn graphs are needed
- Probabilistic methods: better scalability if false positives are acceptable

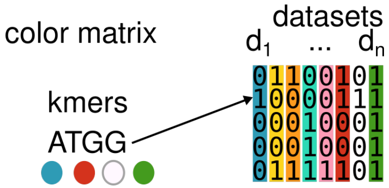
Collections of *k*-mer sets - State of the art



- Different optimizations/features: construction time, space, query speed, dynamicity
- Examples of queries: search of a mutation, alternative splicing, . . .

Collections of *k*-mer sets - Exact methods

Associate *k*-mers to color matrix:

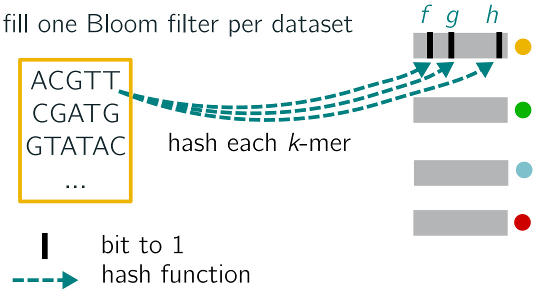


- *k*-mers in SPSS or BWT
- Hash-table or FM-index (BOSS)
- Compression of colored matrix
- Some methods support de Bruijn graph operations

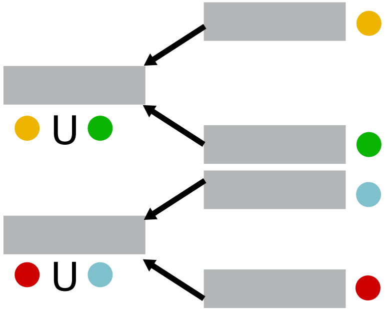
- Examples of exact methods: VARI [Muggli et al. 2017], Mantis [Pandey et al. 2018], BiFrost [Holley & Melsted 2019]

Collections of *k*-mer sets - inexact structures with tree + Bloom Filters

Sequence Bloom Tree (SBT) [Solomon&Kingsford 2016]

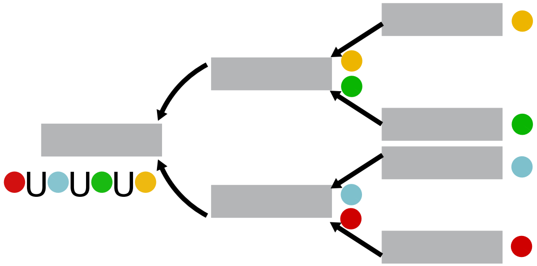


Collections of *k*-mer sets - inexact structures with tree + BF



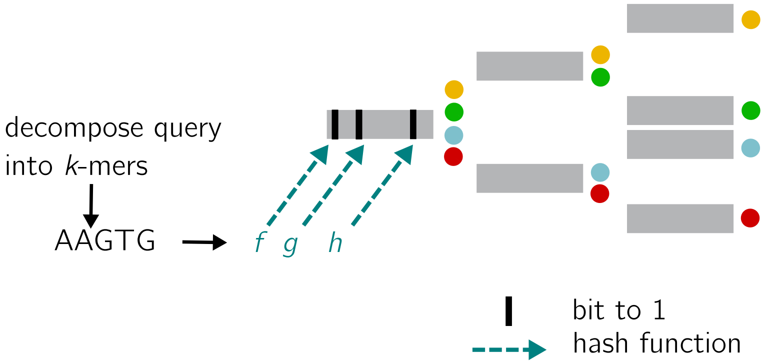
- Same size and hash functions for all filters
- Union : bitwise OR

Collections of *k*-mer sets - Inexact structures with tree + BF

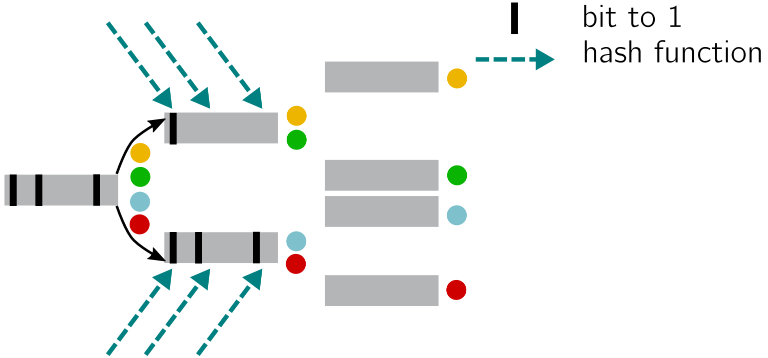


- Upper nodes are more saturated than leaf nodes
- Root node: *k*-mers present in any dataset

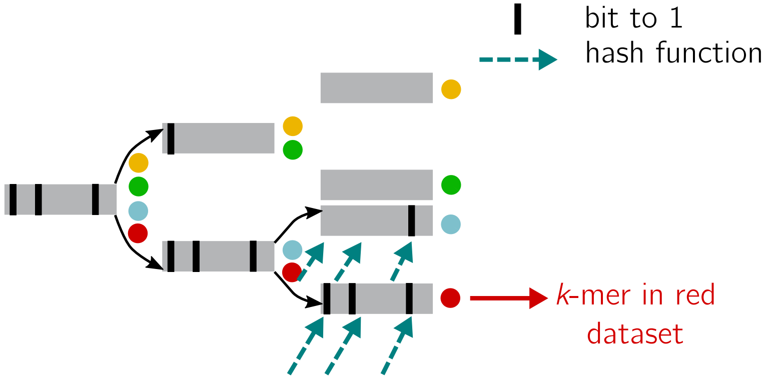
Collections of *k*-mer sets - SBT



Collections of k-mer sets - SBT



Collections of *k*-mer sets - SBT

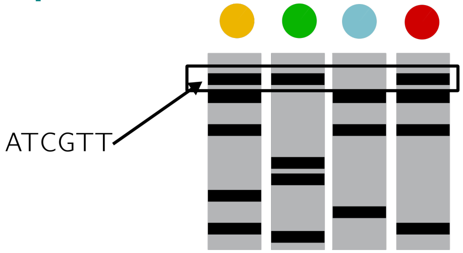


- Expected query: several *k*-mer (amortize FPR)
- $\mathcal{O}\,log(n)$ queries in good cases, worst case in $\mathcal{O}(n)$

Collections of *k*-mer sets - Matrix of BFs

- Query worst case always in $\mathcal{O}(n)$ for these structures
- Random accesses matter

BIGSI [Bradley et al. 2019] and further works: a Bloom filter matrix and inverted index



Collections of k-mer sets - Usage

- Exact methods
  - Need of an extremely precise query/small query ($\sim$ 1 k-mer)
  - A co-assembly graph is useful (variant calling ...)
- Inexact methods
  - A more important need to scale
  - SBTs: similar datasets such as a group of RNA-seq from a cohort + not too many queries
  - BIGSI: genomes of the same species/strain, diverse and large query batches

Examples on $\sim$ 2,500 human RNA-seq (50TB uncompressed)

	inexact structures	exact structures
construction time	2-10h	$\sim$ 20h
index size	$\sim$15 GB	30 GB or more

Sets de sets de *k*-mers - Recent improvements for inexact structures

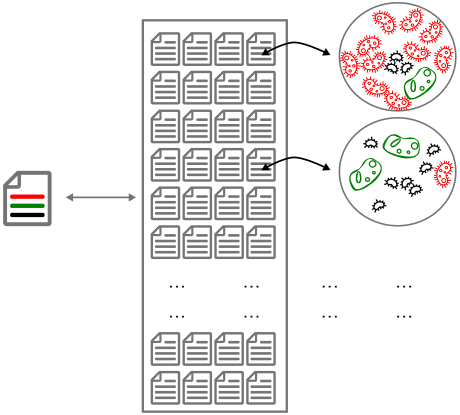
- SBT : leaf clustering, representation of informative bits [Sun et al. 2018, Solomon&Kingsford 2018, Harris&Medvedev 2019]
- BIGSI : Bloom filter folding to adapt to several genome sizes [Bingman et al. 2019]
- Efficient Bloom filter building for these structures [Lemane et al. 2021]
- Improved query speed and FPR for these structures [Robidou & Peterlongo 2021]
- Improved disk footprint + query speed [Marchet & Limasset 2022]

Open questions:

- Combine SPSS and Bloom filter structures
- Can we do better than $\mathcal{O}(n)$ for the query worst case?
- Query expressivity with these structures (*k*-mer abundances?)

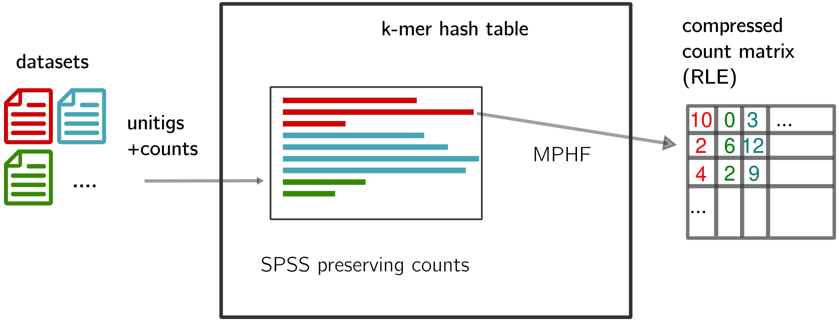
Collections of *k*-mer sets - Quantification

Associate *k*-mers with ~~presence/absence~~ **abundances** in datasets



Collections of *k*-mer sets - large scale abundance index

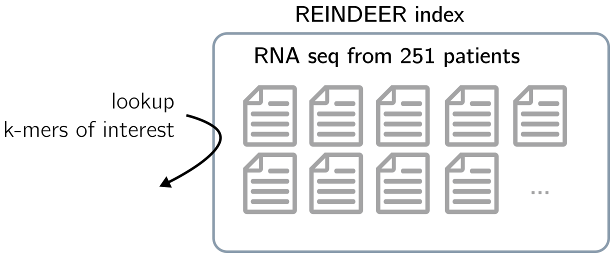
REINDEER [Marchet et al. 2020][7]



[7] and very recently, counting de Bruijn graphs [Karasikov et al. 2021]

Collections of *k*-mer sets - example of application

- Stéphane Pyronnet's team @CRCT Toulouse : **acute myeloid leukemia** (AML)
- An (anonymized) WHO gene is a good prognosis indicator for survival. But why?



- Discovery of new exon-exon junctions (alternative splicing not visible with traditional gene expression), leading to a shorter protein
- mRNA/Protein existence verified with long reads and Western blot
- Discovery of a lncRNA in interaction with the exon junction

Conclusion - large scale *k*-mer data structures

- Currently two visions:
  - With huge computing resources, build very large indexes on servers + APIs
  - "Lightweight" methods for more frugal usages
- A survey on set collections data structures: Marchet C, et al 2019. Data structures based on k-mers for querying large collections of sequencing data sets.

Acknowledgments:

- Paul Bastide
- Team R'n Blood Toulouse: Marina Bousquet, Eulalie Corre, Stéphane Pyronnet
- Team Bio2M Montpellier: Chloé Bessières, Anthony Boureux, Benoit Guibert, Thérèse Commes
- Orsay: Haoliang Xue, Mélina Galopin, Daniel Gautheret
- Team SeqBio Pasteur: Rayan Chikhi
- Team Bonsai Lille: Mikaël Salson, Antoine Limasset, Maël Kerbiriou
- Elsewhere on Earth: Paul Medvedev, Zamin Iqbal, Christina Boucher