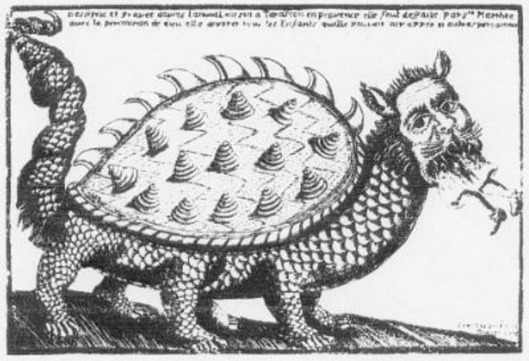
# Indexation de données biologiques : chaque mot compte

Camille Marchet

Univ. Lille, CNRS, CRIStAL, France

camille.marchet@univ-lille.fr
@CamilleMrcht

# Bioinformatics?



**Sequence bioinformatics:** allow the analysis of biological sequences (such as DNA), mostly through text and graph algorithms.

# DNA, RNA, notions you need to brag about Covid-19

**DNA**
aka
"the manual"

...ACTACTCTACTT
TAAGGCAGTAGTG
GATCTACTTACGA
GGGGATTACTAAC
TTGACCTACAGC...

human: **$4.10^9$** characters (=**bases**)

**RNA**
aka
"the blueprint"
or
"the buffer"

.TACGAGGGGA → .KWELLSQ

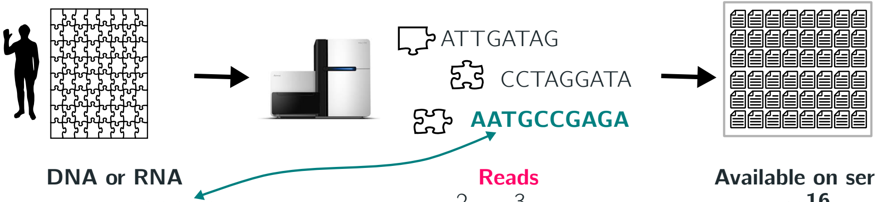
human: **$3.10^5$** different

**protein**
aka
"the effector"

**Texts, alphabet={A,C,G,T}**

# What does our data look like?

We get data through sequencing:



ATTGATAG

CCTAGGATA

**AATGCCGAGA**

**DNA or RNA**

...ATTGATAGCCTA**AATGCCGAGA**GGATA ...

**Reads**

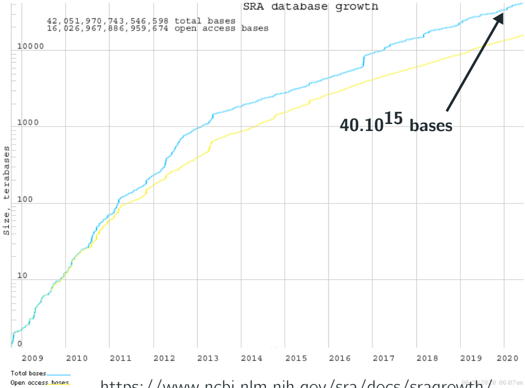
$10^2$ - $10^3$ bases

$\mathbf{10^9}$ fragments per experiment

**Available on servers**

> $\mathbf{4.10^{16}}$ bases

# Indexing available raw sequences (reads) is a challenge



$40.10^{15}$ bases

https://www.ncbi.nlm.nih.gov/sra/docs/sragrowth/

- Petabases of reads stored on servers
- Hardly usable: no possible queries on this data
- Current state-of-the-art: index $10^3$ to $10^6$ datasets for specific queries

## This talk's question: how to find a specific sequence in thousands of datasets or more?

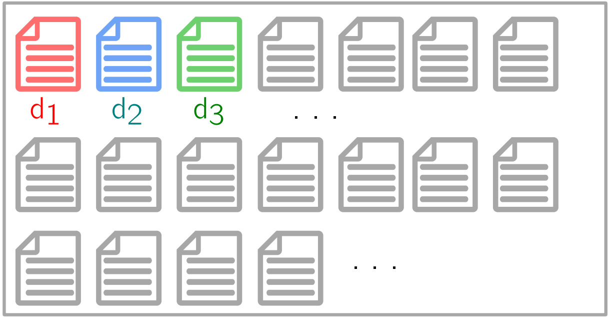
- Different from document retrieval on the web:
    - We want exact matches
    - We want all hits
    - DNA has not a small lexicon such as human languages: many more search terms, the weight grows with each new dataset
- Data-bases, document indexing literature: often too costly, not specific enough to our query type

# Query a sequence to a collection of datasets

a set of datasets $\{d_1, d_2, \ldots d_n\}$
(reads multisets)

query sequence

...ATTACGTAGTA...



**return** all $d_i$'s where the query occurs

# Use words: *k*-mers

Decompose sequence to ***k-mers***

```
TGATGATTAGC
TGATG
 GATGA
  ATGAT
   TGATT
     ...
```

*k* size order: 20-100

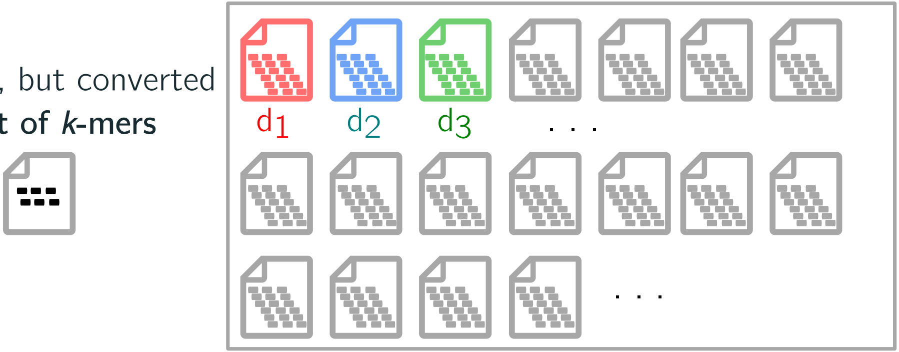
Main advantage:

- lower orders of magnitude to index
  e.g.: $10^9$ reads per dataset in $10^3$ datasets becomes $10^9$ *k*-mers in total
- the query remains quite specific with long enough *k*-mers

# Use words: *k*-mers

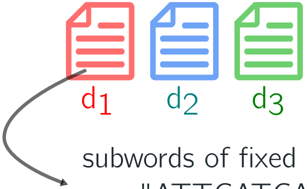
same datasets, but seen as ***k*-mer sets**

same query, but converted to a **set of *k*-mers**



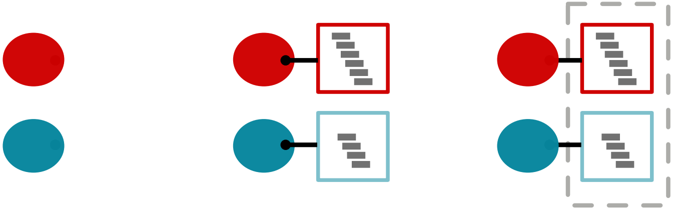
report $d_i$'s **containing sufficiently enough (≥t) *k*-mers from the query** [Solomon & Kingsford '16]

# Summary

| | Nature | Order of magnitude |
|---|---|---|
| **Datasets** | biological samples, made of text<br><br> | $10^3$ datasets,<br>several Terabytes of input |
| **Elements** | subwords of fixed length<br>e.g. "ATTGATCA" | $>10^9$ elements |
| **Query** | set of subwords of fixed length<br>e.g. {"ATTGATCA", "TTTTTTCA"} | $> 10^3$ queries, containing<br>10/100/1,000 subwords |

# State of the art

*k*-mer aggregative methods:



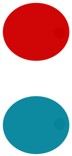
step 0: datasets (called colors)

step 1: represent each dataset

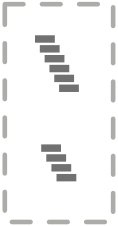
step 2: query the union

# State of the art

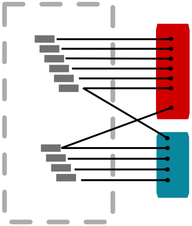
color aggregative methods:



step 0: datasets (called colors)



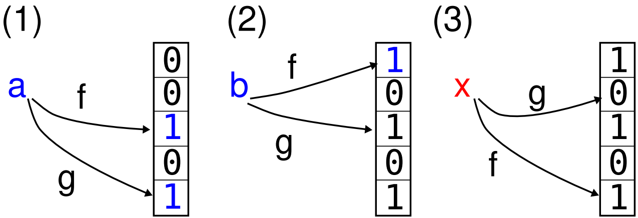
step 1: index the intersection



step 2: associate to colors

# Bloom filters [Bloom ’70]

$S_1=\{a,b\}$, $x \notin S_1$

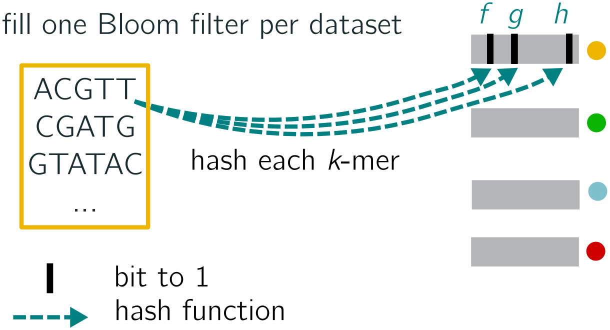


f(x) AND g(x)=0

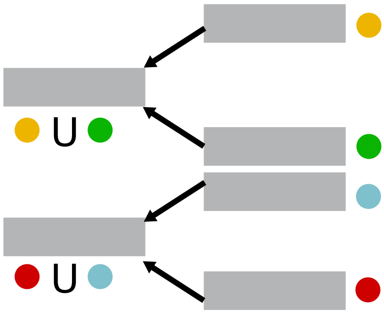
⇒ x not in $S_1$

- Probabilistic way to represent a set
- Controled false positive rate
- No false negative

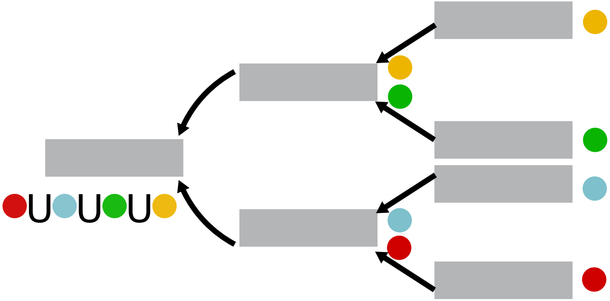
# $K$-mer aggregative method: the Sequence Bloom Tree (SBT)



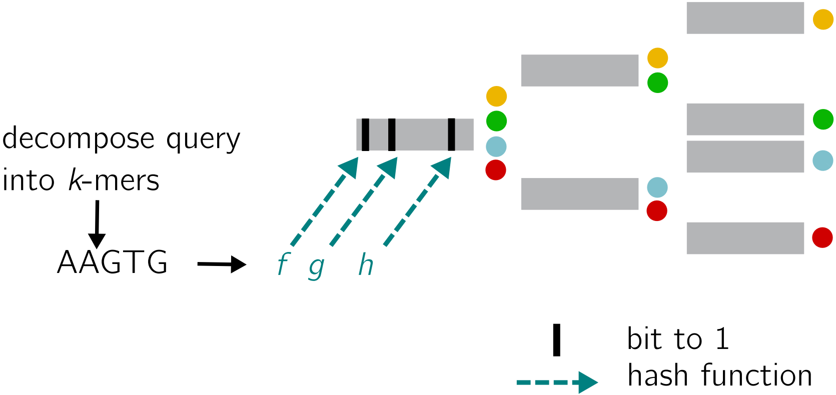
# Sequence Bloom Tree (SBT) construction



# Sequence Bloom Tree (SBT) construction



# Query the Sequence Bloom Tree (SBT)



decompose query into *k*-mers

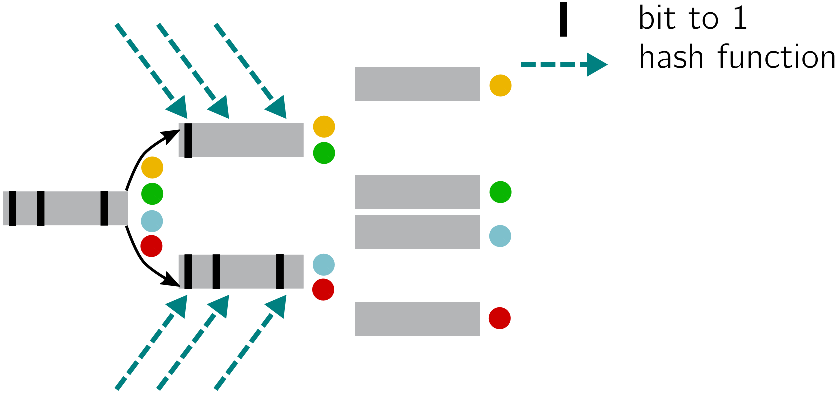
AAGTG

*f* *g* *h*

❙ bit to 1

⇢ hash function

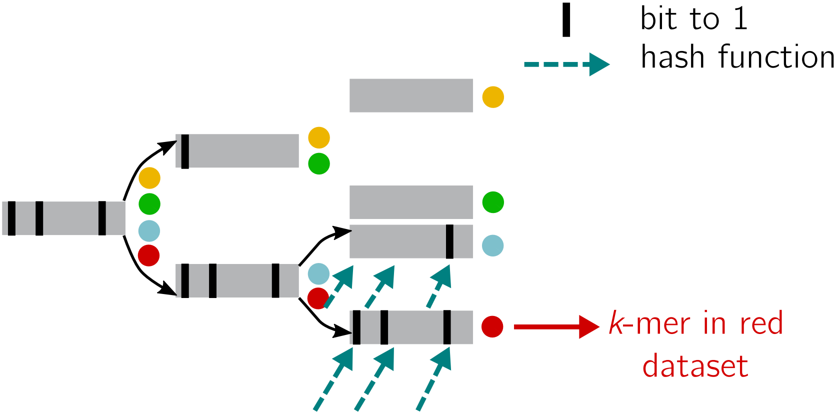
# Query the Sequence Bloom Tree (SBT)



▌ bit to 1

hash function

# Query the Sequence Bloom Tree (SBT)



▌ bit to 1

- - -▶ hash function

*k*-mer in red dataset

# Query the Sequence Bloom Tree (SBT)

- Repeat for each *k*-mer of the query
- If enough *k*-mers from the query fall in a dataset, report this dataset

- Hash-based: very quick response
- The false positive rate for a whole query decreases as the number of *k*-mers increases (see Solomon & Kingsford '16 for an analysis)

# Sequence Bloom Tree (SBT)

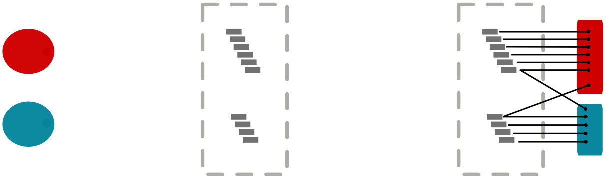
## Performances (example)

- Input: raw data is 5TB, 2,500 datasets with 3.7 billion *k*-mers
- Latest SBT index size in RAM: 200 GB

## Improvements

- Clustering of the leaves [Sun et al. '18]
- Remove redundancy in internal nodes [Solomon and Kingsford '18, Sun et al. '18, Harris and Medvedev '19]
- Forest instead of a tree [Harris and Medvedev '19]
- Same paradigm but using matrix-like structures instead of trees [Bradley et al. '19, Bingman et al. '19]

# Color aggregative methods



step 0: datasets
(called colors)

step 1: index
the intersection

step 2: associate
to colors

efficient k-mer representation:
active research field
[Brinda et al. '20,
Rahman & Medvedev '20,
Marchet et al. '20]

several hash-based solutions:
- Minimal perfect hashing [Marchet et al. '20,
Almodaresi et al. '19,
Holley and Melsted '19]
- Othello hashing [Yu et al. '18]
- Counting quotient filters [Pandey et al. '18]

# Open problems

- Record other information than *k*-mer presence/absence in a dataset (quantification, read presence instead of word presence, . . . )
- Represent efficiently a set of *k*-mers
- Plethora of propositions but can we find a ”holy grail” data-structure?

→ Further questions in Antoine’s talk
→ Feel free to write for internship inquiries:
camille.marchet@univ-lille.fr
🐦 @CamilleMrcht

# Bibliography

**About *k*-mer representation:**

- Representation of k-mer sets using spectrum-preserving string sets, Rahman et al., ’20
- `https://kamimrcht.github.io/webpage/tigs.html`

**About SBT:**

- Fast search of thousands of short-read sequencing experiments, Solomon and Kingsford, ’16
- Improved representation of sequence bloom trees, Harris and Medvedev, ’18

**About hashing in color aggregative methods:**

- Minimal Perfect hashing: Fast and scalable minimal perfect hashing for massive key sets, Limasset et al., ’17
- Counting quotient filters: A General-Purpose Counting Filter: Making Every Bit Count, Pandey et al., ’17
- Othello hashing: `https://kamimrcht.github.io/webpage/othello.html`

**Recent survey:**

- Data structures based on k-mers for querying large collections of sequencing datasets, Marchet et al., ’20