Scalable indexing and mapping of long seqs through locally consistent phrases

Igor MARTAYAN, Rob PATRO, Camille MARCHET

DSB 2026, February 19th, Venice

Context: accurate long reads enable new methods

Sequencing data keeps getting longer and more accurate

Recent trend: working on sequence of “minimizers”
(smaller sequences on a bigger alphabet)

- Minimizer-space dBG (Ekim et al. 21)
- Mapquik (Ekim et al. 23)
- U-index (Ayad et al. 25)

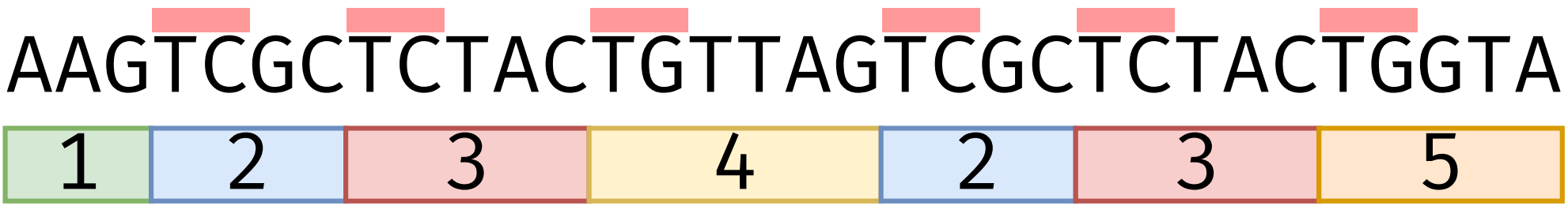
Can we adapt these ideas into a lossless framework?

From sampling anchors to parsing phrases

Start by sampling anchors (e.g. minimizers) in your sequence

From sampling anchors to parsing phrases

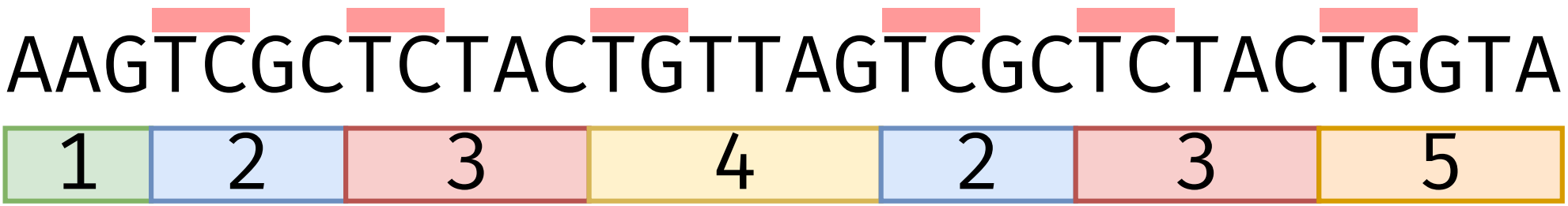
Start by sampling anchors (e.g. minimizers) in your sequence



then look at the phrases *between* anchors

From sampling anchors to parsing phrases

Start by sampling anchors (e.g. minimizers) in your sequence



then look at the phrases *between* anchors

Local consistency

same context ⇒ same anchors ⇒ same phrases

A global framework based on consistent phrases

1. **parse** the sequence into locally consistent phrases
2. **fingerprint** the phrases (e.g. with their lex. rank, hash...)
3. **index** the sequence of fingerprints

the sequence of fingerprints is much shorter (# phrases),
but has a larger alphabet (# distinct fingerprints)

Parsing

Fingerprinting

Indexing

Querying

Different ways to select the anchors

We'll focus on sampling *k*-mers based on their (canonical) **hash**

Sampling parameter *p*

Expected distance between selected *k*-mers

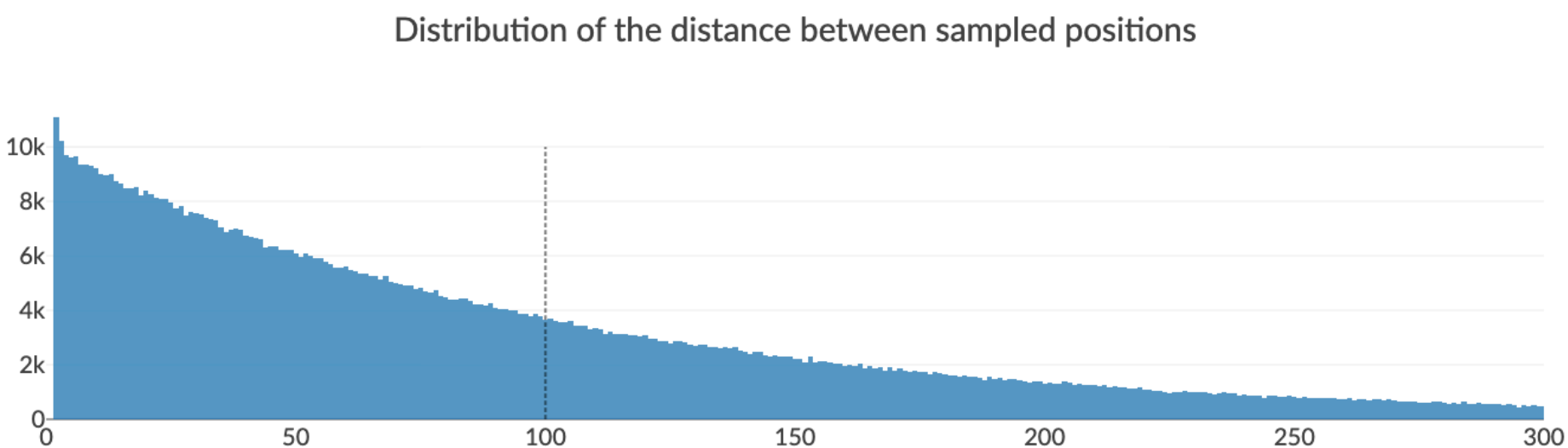
- **context-free**: e.g. $h = 0 \bmod p$ or $h \leq \frac{\texttt{maxint}}{p}$ (prefix-free parsing)

$\rightarrow$ simple and fast, but the distance is *unbounded*

- **context-based**: e.g. minimizer schemes, syncmers...

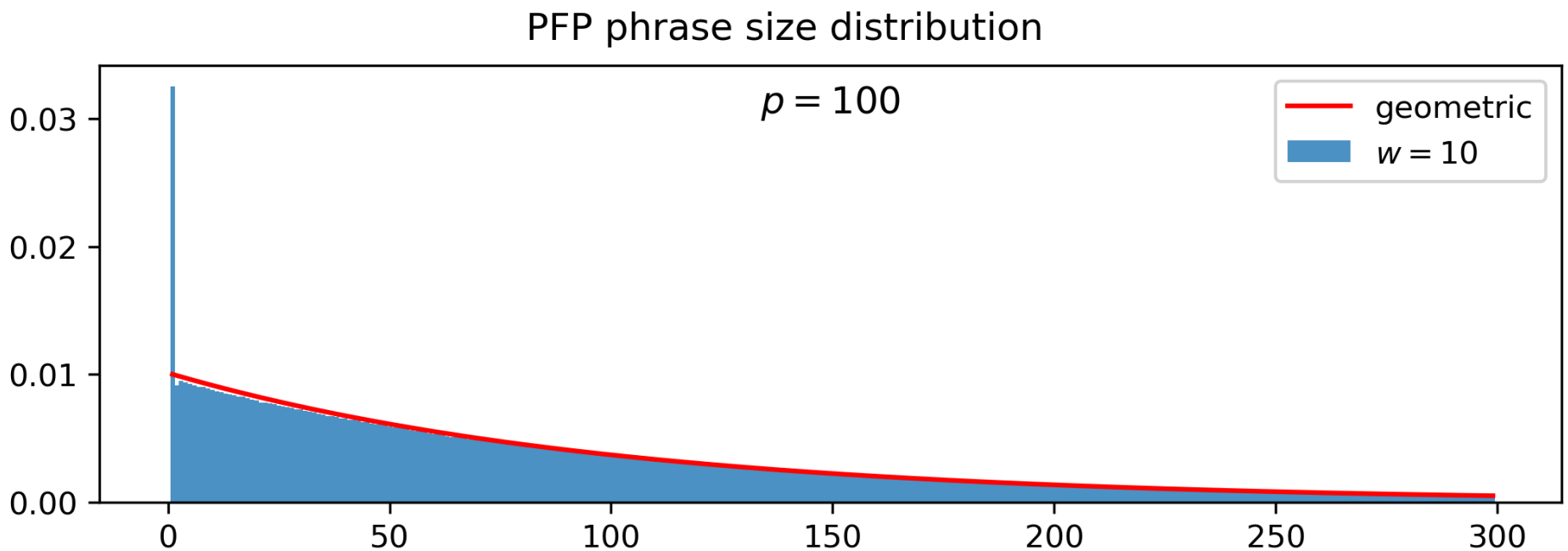
$\rightarrow$ more involved, but ensure a *window guarantee*

A look at prefix-free parsing (PFP)



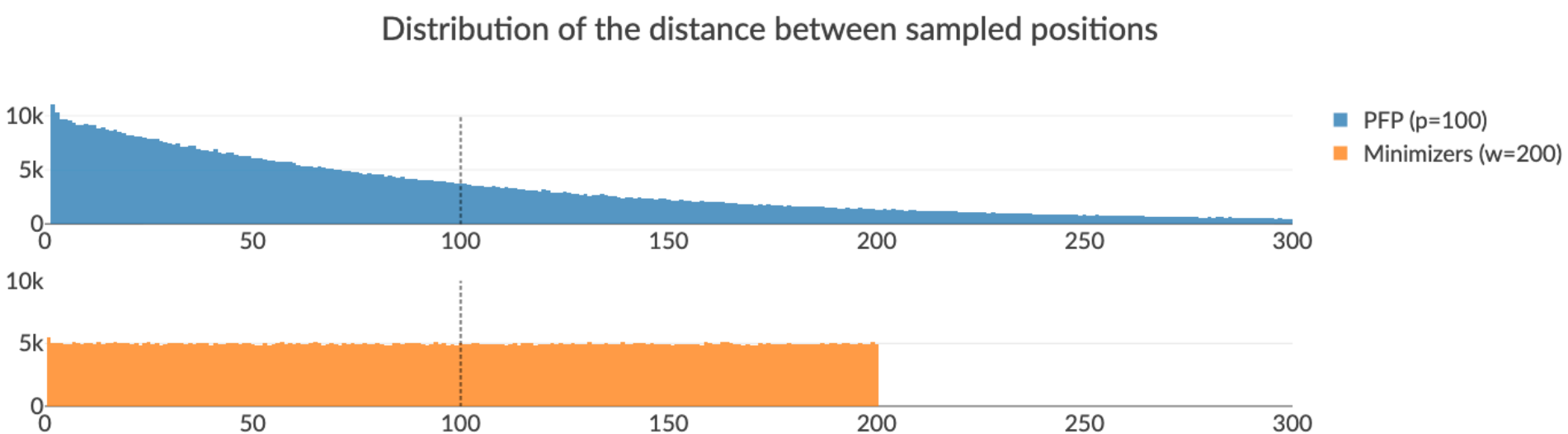
Context-free anchors produce a ~geometric distribution

Sidenote: Biased hash (NtHash) → biased distribution



Experiments available at github.com/imartayan/pfp-distribution

Enforcing an upper-bound with minimizers

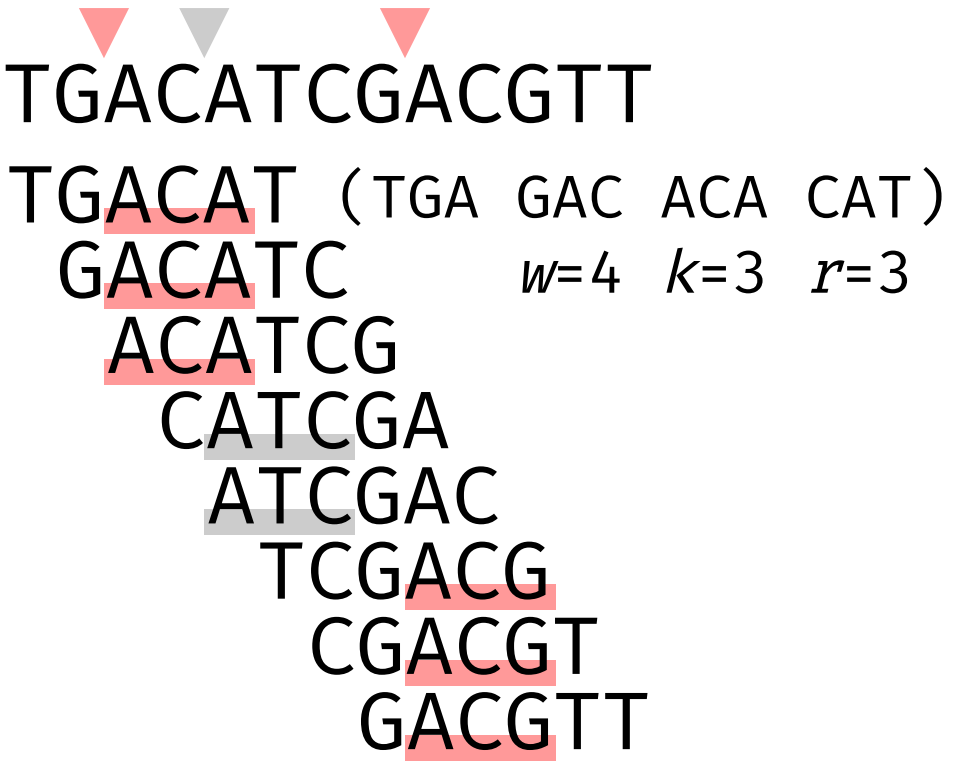


Random minimizers produce a ~uniform distribution

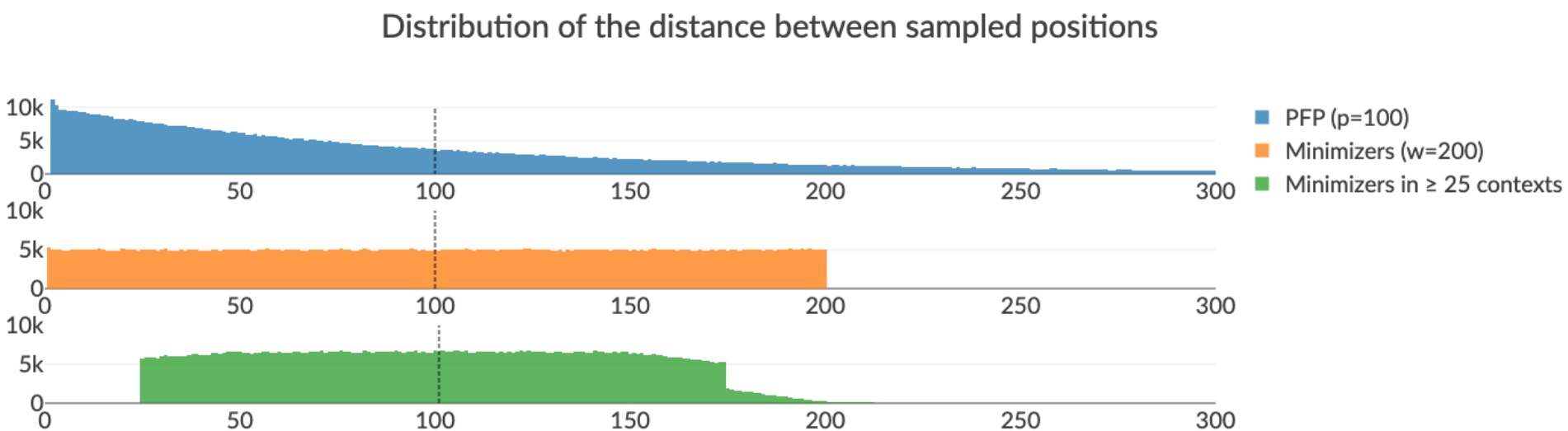
Enforcing a lower bound with *robust* minimizers

Given a robustness factor r, only keep minimizers that are selected in a least r windows.

$\Rightarrow$ at least r positions apart



Enforcing a lower bound with *robust* minimizers



Open question: can we enforce a lower *and* an upper bound?

Parsing

Fingerprinting

Indexing

Querying

Representing phrases by their fingerprints

- Instead of storing the raw sequences of each phrase, we can represent them by small fingerprints (e.g. 64-bit hashes)
- If we use a rolling hash, this step can be merged with parsing!
- Enables fast exact matches, but not fine-grained comparison

What if we want to match phrases within small edit distance?

Using small sketches for approximate matching

Idea: compute sketches for each phrase to estimate similarity

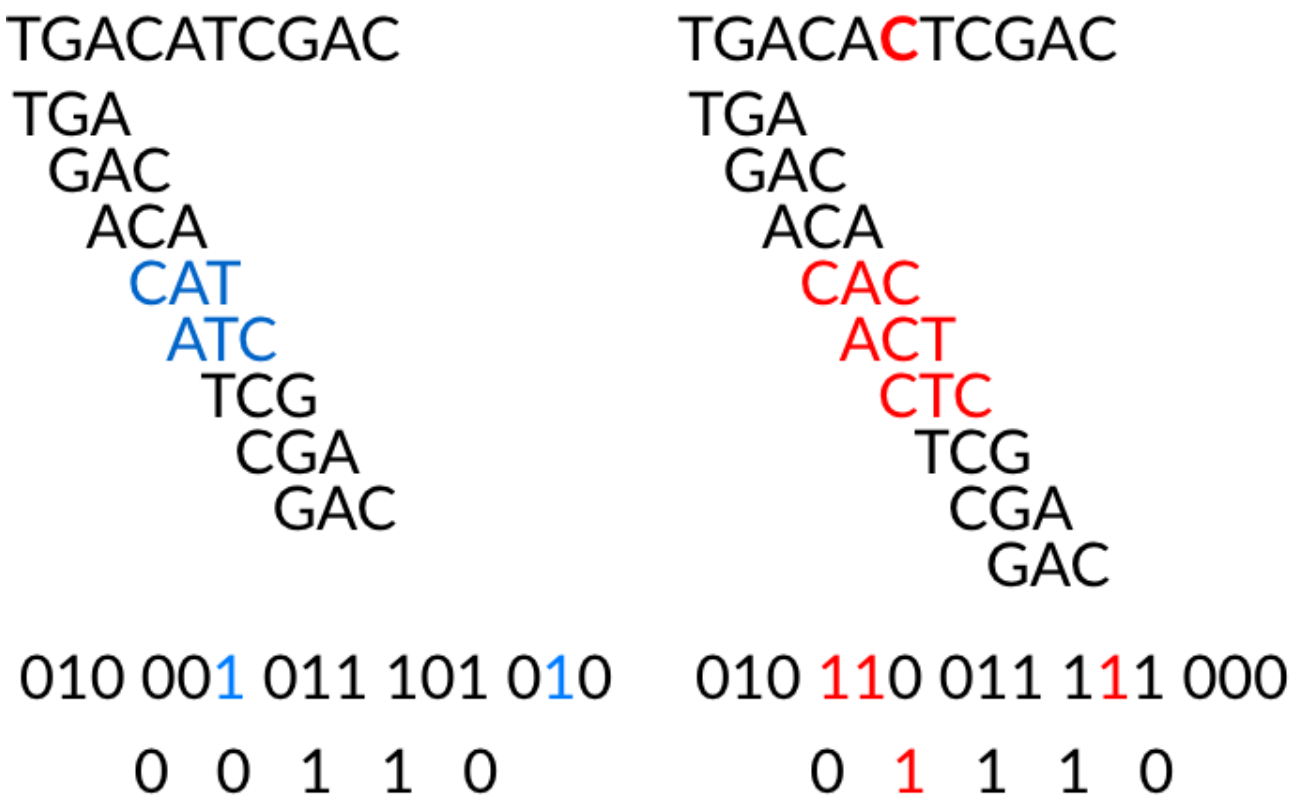
One approach:

- store the k-mers of a phrase Bloom filter (k should be small!)
- build a small sketch by averaging blocks from the Bloom filter

quantized BF of similar phrases have a low Hamming distance

(other ideas worth exploring: MinHash, DotHash)

Quantized Bloom Filter



Parsing

Fingerprinting

Indexing

Querying

Overview of the (uncompressed) index

Component	Data structure	Size estimation (on CHM13v2, p=100)
Phrase positions	Elias-Fano	5 MB
Phrase orientations	Bitvector	0.5 MB
Phrase fingerprints	Array	15 MB
Index	Suffix array	15 MB

Total size: ~35 MB for a human genome (3.2 Gbp)

How fast is it? (on my laptop, using 8 threads)

	Incremental steps	Throughput
1.	Hashing k-mers	15 Gbp/s
2.	Sampling minimizers*	5 Gbp/s
3.	Hashing phrases	5 Gbp/s
4.	Indexing fingerprints	1.5 Gbp/s

$\rightarrow$ indexing a human genome takes ~2s

*using github.com/rust-seq/simd-minimizers

Towards a compressed version of the index (not implemented yet)

If the text is repetitive, we can store the index in $O(\#\text{BWT runs})$ using different data structures:

- any compressed representation of the phrases supporting random access, e.g. **relative Lempel-Ziv** (Kuruppu et al. 10)
- the **suffixient array** of the fingerprints (Cenzato et al. 24)

Parsing

Fingerprinting

Indexing

Querying

Using this index for mapping (WIP)

Heuristics:

1. look for exact matches on the fingerprints
   using the suffix(ient) array (similar to STAR)

Using this index for mapping (WIP)

Heuristics:

1. look for exact matches on the fingerprints using the suffix(ient) array (similar to STAR)
2. try to extend them as much as possible with approximate matches (with few errors)

Using this index for mapping (WIP)

Heuristics:

1. look for exact matches on the fingerprints
   using the suffix(ient) array (similar to STAR)
2. try to extend them as much as possible
   with approximate matches (with few errors)
3. chain close matches that have the same orientation

Take-home messages

- very flexible framework (parse / fingerprint / index)
- supports both exact and approximate queries
- vectorized implementation
- (optional) compressed space

What's next?

- finish the mapper
- experiment with the different parameters (p, r, k...)
- implement compressed index

Take-home messages

- very flexible framework (parse / fingerprint / index)
- supports both exact and approximate queries
- vectorized implementation
- (optional) compressed space

Thank you!

What's next?

- finish the mapper
- experiment with the different parameters (p, r, k...)
- implement compressed index

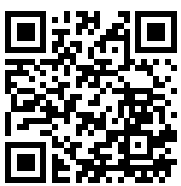
Rust crates for vectorized sequence processing 🦀

Helicase: vectorized parsing & bitpacking

github.com/imartayan/helicase

Seq-hash: vectorized rolling hash

github.com/rust-seq/seq-hash



SIMD minimizers & syncmers

github.com/rust-seq/simd-minimizers