Using necklaces to build a locality-preserving and dynamic index for *k*-mers

Igor Martayan, Bastien Cazaux, Camille Marchet, Antoine Limasset

December 14, 2023

Seminar on Lyndon words — Rouen

Université de Lille
cnrs
CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

DNA Sequencing & Tokenization with k-mers

DNA samples 🧬 $\longrightarrow$

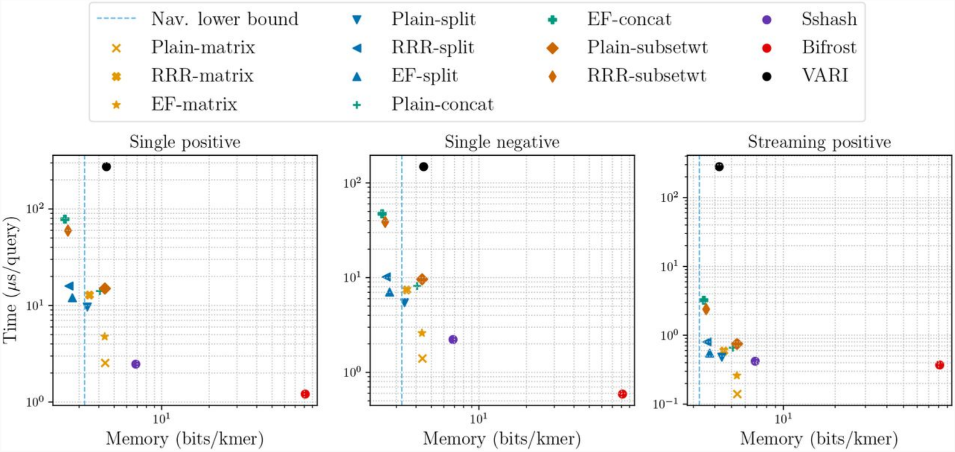
$\longrightarrow$ CTGAAATG...

We typically index the words of size k (k-mers) instead of the sequence itself.

In practice, we usually consider $k \leqslant 63$ so that each k-mer fits inside a machine word.

```
CTGAA
 TGAAA
  GAAAT
   AAATG
```

Plenty of compact data structures for storing k-mers ...but most of them are static



Query time and memory usage of some efficient data structures, taken from [Alanko et al. 22]

Revisiting a simple idea: *k*-mers as a sparse set of integers

[Conway & Bromage 11]

- we can see *k*-mers as integers in $[\![4^k]\!]$
  A $\rightarrow$ 00   C $\rightarrow$ 01   G $\rightarrow$ 10   T $\rightarrow$ 11
- since they're usually very sparse, we can use a sparse bitvector to store them

Limitations

- it's not really dynamic
- it's not cache-efficient
  - index(ATAACGCCA) = 49,556
  - index(TAACGCCAT) = 198,227

$\rightarrow$ average distance of $4^k/2$

REVISITING A SIMPLE IDEA: K-MERS AS A SPARSE SET OF INTEGERS

[Conway & Bromage 11]

- we can see k-mers as integers in $[\![4^k]\!]$
  A $\rightarrow$ 00   C $\rightarrow$ 01   G $\rightarrow$ 10   T $\rightarrow$ 11
- since they're usually very sparse, we can use a sparse bitvector to store them

Limitations

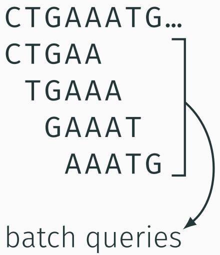
- it's not really dynamic
- it's not cache-efficient
  - index(ATAACGCCA) = 49,556
  - index(TAACGCCAT) = 198,227

$\rightarrow$ average distance of $4^k/2$

How can we improve this approach?

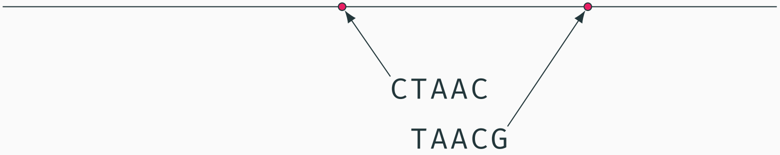
Wish list for an ideal data structure

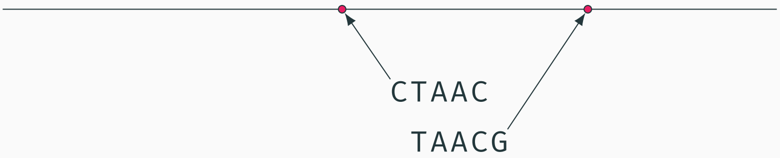
- space-efficient: few bits / k-mer
- dynamic: support insertion and deletion after construction
- efficient queries:
  - membership
  - enumeration
  - insertion
  - deletion
- locality-preserving: reduce cache misses when querying consecutive k-mers



Preserving locality with necklaces

A locality-preserving encoding of *k*-mers

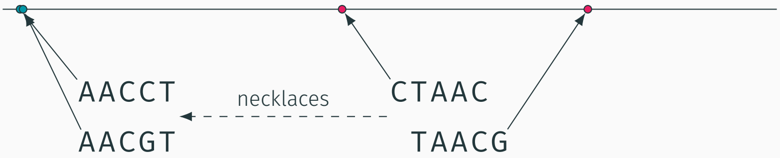




Alternative encoding based on necklaces

The necklace of x is its smallest cyclic rotation $\langle x \rangle = \min_{0 \leqslant i < k} x^{(i)}$

A locality-preserving encoding of k-mers



Alternative encoding based on necklaces

The necklace of x is its smallest cyclic rotation $\langle x \rangle = \min_{0 \leqslant i < k} x^{(i)}$

- $x \mapsto (\langle x \rangle,$ rotation index) is a reversible transformation
- necklaces of consecutive k-mers share long prefixes

A CLOSER LOOK AT THE LOCALITY OF NECKLACES

AACGTCATCTCTCATTCT**GG**TCGTTCTTCCT

AACGTCATCTCTCATTCTG**TT**CGTTCTTCCT

AACGTCATCTCTCATTCTGT**GC**GTTCTTCCT

AACGTCATCTCTCATTCTGTG**AG**TTCTTCCT

AACGTCATCTCTCATTCTGTGA**CT**TCTTCCT

AACGTCATCTCTCATTCTGTGAC**AT**CTTCCT

AACGTCATCTCTCATTCTGTGACA**CC**TTCCT

AACGTCATCTCTCATTCTGTGACAC**GT**TCCT

AACGTCATCTCTCATTCTGTGACACG**CT**CCT

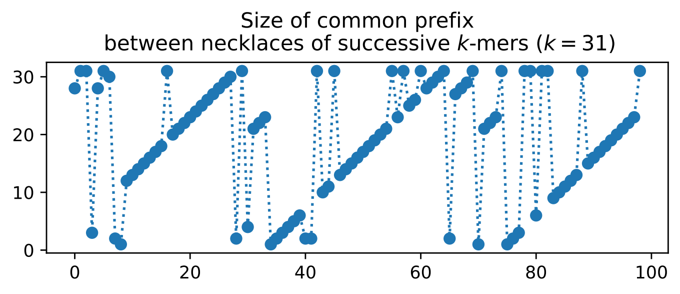
AACGTCATCTCTCATTCTGTGACACGC**AC**CT

AACGTCATCTCTCATTCTGTGACACGCA**GC**T

AACGTCATCTCTCATTCTGTGACACGCAG**GT**

AACGTCATCTCTCATTCTGTGACACGCAGG**G**

ACACGCAGGG**TA**CGTCATCTCTCATTCTGTG



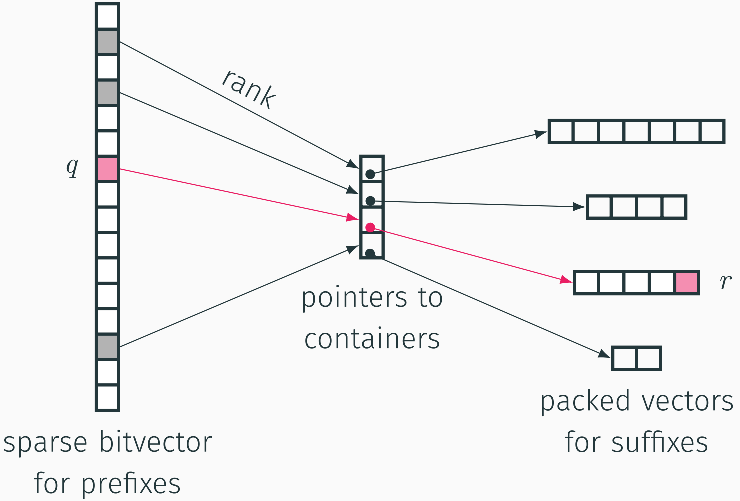
Practical use of necklaces

Overview of our data structure (*CBL*)

Quotiented
data structure

Query x :

1. compute $\langle x \rangle$
2. split $\langle x \rangle$ as $q \,||\, r$
3. look for (q, r)



Basic approach: compute every cyclic rotation and select the smallest in $\mathcal{O}(k)$.
$\rightarrow \mathcal{O}(nk)$ for n queries

Better approach: amortize the computation cost for consecutive queries.

Key observation

Given a fixed m, if $\langle x \rangle$ does not start at one of the $m-1$ last positions of x, its prefix of size m is the smallest factor of size m in x.

Good news: we can keep track of the smallest factors of size m in $\mathcal{O}(1)$ amortized time using a monotone queue.

Faster necklace computation

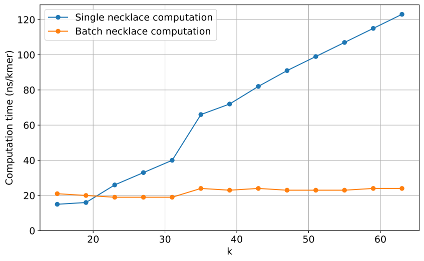
Only consider the cyclic rotations that start:

- at one of the smallest factors of size m
- at one of the $m-1$ last positions

Useful property [Zheng et al. 20]

Assuming $m = \Omega(\log k)$, the probability that a k-mer contains duplicate m-mers is $o(1/k)$.

By choosing $m = \Theta(\log k)$,
the smallest factor of size m is unique w.h.p.
$\rightarrow \mathcal{O}(nm) = \mathcal{O}(n \log k)$ for n queries (on average)



Densifiying the space of necklaces

Densifiying the space of necklaces by ranking

The number of necklaces of size k on an alphabet with σ letters is

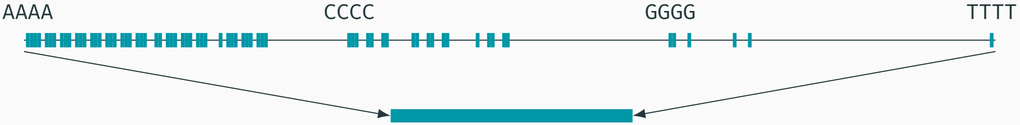
$$N(k) = \frac{1}{k}\sum_{d|k}\varphi\left(\frac{k}{d}\right)\sigma^d \sim \frac{\sigma^k}{k}$$

so only a fraction $\frac{1}{k}$ of the universe is actually used

The number of necklaces of size k on an alphabet with σ letters is

$$N(k) = \frac{1}{k}\sum_{d|k} \varphi\left(\frac{k}{d}\right)\sigma^d \sim \frac{\sigma^k}{k}$$

so only a fraction $\frac{1}{k}$ of the universe is actually used



Ranking: given a necklace $\langle x \rangle$, find i s.t. $\langle x \rangle$ is the i-th smallest necklace of size k

We can compute the rank in $\mathcal{O}(k^2)$ time [Sawada & Williams 17]

Tradeoff: better locality + compression vs $\mathcal{O}(k^2)$ queries

Ranking in $\mathcal{O}(k^2)$ is generally too expensive for our use case, but it might be faster to rank necklaces of consecutive k-mers.

Since most necklaces of consecutive words share the same starting position, they only differ by a single letter.

AACGTCATCTCTCATTCTG**G**TCGTTCTTCCT
AACGTCATCTCTCATTCTG**T**TCGTTCTTCCT

Formulation in the binary case ($\sigma = 2$)

How does the rank of $\langle x \rangle$ change if we flip its i-th bit?

Conclusion

Indexing k-mers with their necklaces:

- preserves locality
- improves compression
- fits in well with a quotiented data structure
- combines easily with dynamic operations

Future questions:

- What is the average distance between necklaces of consecutive k-mers?
- Can we rank necklaces in subquadratic time?
- Can we accelerate ranking for necklaces of consecutive k-mers?

Take-home messages & Open questions

Indexing k-mers with their necklaces:

- preserves locality
- improves compression
- fits in well with a quotiented data structure
- combines easily with dynamic operations

Future questions:

- What is the average distance between necklaces of consecutive k-mers?
- Can we rank necklaces in subquadratic time?
- Can we accelerate ranking for necklaces of consecutive k-mers?

Thank you!

Alanko, Jarno N, Simon J Puglisi & Jaakko Vuohtoniemi (2022). **"Succinct k-mer sets using subset rank queries on the spectral burrows-wheeler transform".** In: *bioRxiv*, pp. 2022–05.

Conway, Thomas C & Andrew J Bromage (2011). **"Succinct data structures for assembling large genomes".** In: *Bioinformatics* 27.4, pp. 479–486.

Sawada, Joe & Aaron Williams (2017). **"Practical algorithms to rank necklaces, Lyndon words, and de Bruijn sequences".** In: *Journal of Discrete Algorithms* 43, pp. 95–110.

Zheng, Hongyu, Carl Kingsford & Guillaume Marçais (2020). **"Improved design and analysis of practical minimizers".** In: *Bioinformatics* 36.Supplement_1, pp. i119–i127.