Conway-Bromage-Lyndon (CBL):

an exact, dynamic representation of k-mer sets

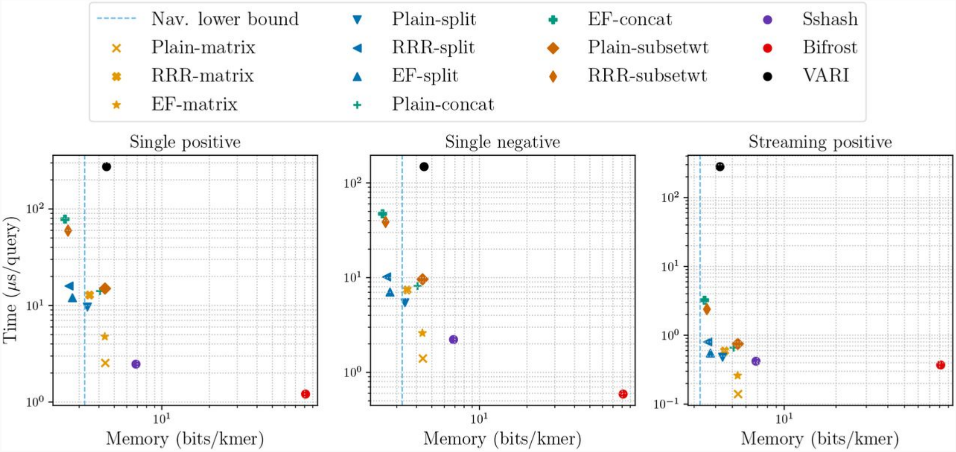
Igor Martayan, Bastien Cazaux, Antoine Limasset & Camille Marchet

March 15, 2024

DSB 2024 — Montpellier

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

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]

Goal: designing a dynamic index of k-mers
with fast queries and relatively good compression

- membership
- enumeration
- insertion
- deletion
- set operations ($\cup, \cap, \setminus$)

Starting from a simple idea: k-mers as a sparse set of integers

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

Limitations:

- difficult to compress
  (especially if it's dynamic)
- not cache-efficient
  id(ATGGCA) $\ll$ id(TGGCAT)
  (average distance of $4^k/3$)

Starting from a simple idea: k-mers as a sparse set of integers

- 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 store them in a sparse bitvector (as in [Conway & Bromage 11])

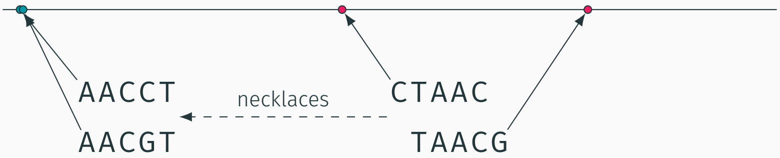
Limitations:

- difficult to compress (especially if it's dynamic)
- not cache-efficient
  id(ATGGCA) $\ll$ id(TGGCAT)
  (average distance of $4^k/3$)

What if we changed our representation of k-mers?

The necklace transformation

Necklace transformation of k-mers



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

To make this transformation reversible, keep track of the rotation index

$$x \longmapsto (\langle x \rangle, \text{rotation index})$$

Necklaces of consecutive k-mers share long prefixes

k-mer view

GTCGTTCTTCCTAACGTCATCTCTCATTCTG

TCGTTCTTCCTAACGTCATCTCTCATTCTGT

CGTTCTTCCTAACGTCATCTCTCATTCTGTG

GTTCTTCCTAACGTCATCTCTCATTCTGTGA

TTCTTCCTAACGTCATCTCTCATTCTGTGAC

TCTTCCTAACGTCATCTCTCATTCTGTGACA

CTTCCTAACGTCATCTCTCATTCTGTGACAC

TTCCTAACGTCATCTCTCATTCTGTGACACG

TCCTAACGTCATCTCTCATTCTGTGACACGC

CCTAACGTCATCTCTCATTCTGTGACACGCA

CTAACGTCATCTCTCATTCTGTGACACGCAG

TAACGTCATCTCTCATTCTGTGACACGCAGG

AACGTCATCTCTCATTCTGTGACACGCAGGG

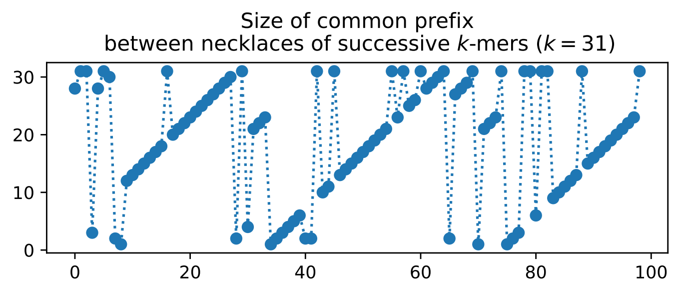
ACGTCATCTCTCATTCTGTGACACGCAGGGT

NECKLACES OF CONSECUTIVE *k*-MERS SHARE LONG PREFIXES

necklace view	*k*-mer view
AACGTCATCTCTCATTCTG GTCGTTCTTCCT	GTCGTTCTTCCTAACGTCATCTCTCATTCTG
AACGTCATCTCTCATTCTGT TCGTTCTTCCT	TCGTTCTTCCTAACGTCATCTCTCATTCTGT
AACGTCATCTCTCATTCTGTG CGTTCTTCCT	CGTTCTTCCTAACGTCATCTCTCATTCTGTG
AACGTCATCTCTCATTCTGTGA GTTCTTCCT	GTTCTTCCTAACGTCATCTCTCATTCTGTGA
AACGTCATCTCTCATTCTGTGAC TTCTTCCT	TTCTTCCTAACGTCATCTCTCATTCTGTGAC
AACGTCATCTCTCATTCTGTGACA TCTTCCT	TCTTCCTAACGTCATCTCTCATTCTGTGACA
AACGTCATCTCTCATTCTGTGACAC CTTCCT	CTTCCTAACGTCATCTCTCATTCTGTGACAC
AACGTCATCTCTCATTCTGTGACACG TTCCT	TTCCTAACGTCATCTCTCATTCTGTGACACG
AACGTCATCTCTCATTCTGTGACACGC TCCT	TCCTAACGTCATCTCTCATTCTGTGACACGC
AACGTCATCTCTCATTCTGTGACACGCA CCT	CCTAACGTCATCTCTCATTCTGTGACACGCA
AACGTCATCTCTCATTCTGTGACACGCAG CT	CTAACGTCATCTCTCATTCTGTGACACGCAG
AACGTCATCTCTCATTCTGTGACACGCAGG T	TAACGTCATCTCTCATTCTGTGACACGCAGG
AACGTCATCTCTCATTCTGTGACACGCAGGG	AACGTCATCTCTCATTCTGTGACACGCAGGG
ACACGCAGGGT ACGTCATCTCTCATTCTGTG	ACGTCATCTCTCATTCTGTGACACGCAGGGT

necklace view

AACGTCATCTCTCATTCTG GTCGTTCTTCCT
AACGTCATCTCTCATTCTGT TCGTTCTTCCT
AACGTCATCTCTCATTCTGTG CGTTCTTCCT
AACGTCATCTCTCATTCTGTGA GTTCTTCCT
AACGTCATCTCTCATTCTGTGAC TTCTTCCT
AACGTCATCTCTCATTCTGTGACA TCTTCCT
AACGTCATCTCTCATTCTGTGACAC CTTCCT
AACGTCATCTCTCATTCTGTGACACG TTCCT
AACGTCATCTCTCATTCTGTGACACGC TCCT
AACGTCATCTCTCATTCTGTGACACGCA CCT
AACGTCATCTCTCATTCTGTGACACGCAG CT
AACGTCATCTCTCATTCTGTGACACGCAGG T
AACGTCATCTCTCATTCTGTGACACGCAGGG
ACACGCAGGGT ACGTCATCTCTCATTCTGTG



Quickly computing streams of necklaces

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

Better: amortize the computation for consecutive k-mers.

Key observation

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

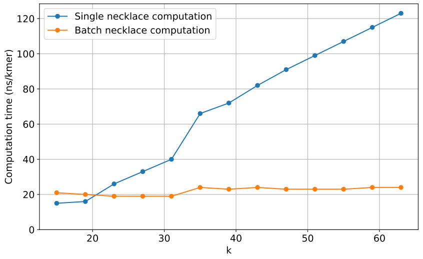
Only consider the cyclic rotations that start:

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

Useful property [Zheng et al. 20]

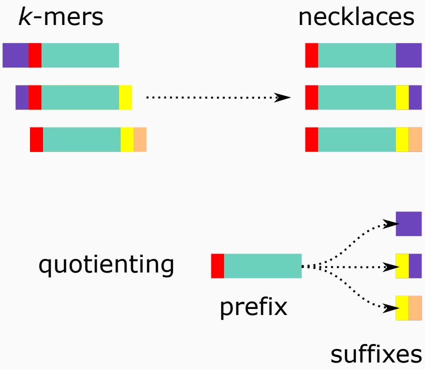
Assuming $m = \Omega(\log k)$, the probability that a k-mer contains duplicate m-mers is $o(1/k)$.

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 necklaces (on avg)



Design of the data structure

Quotienting the prefixes of necklaces



Quotienting:

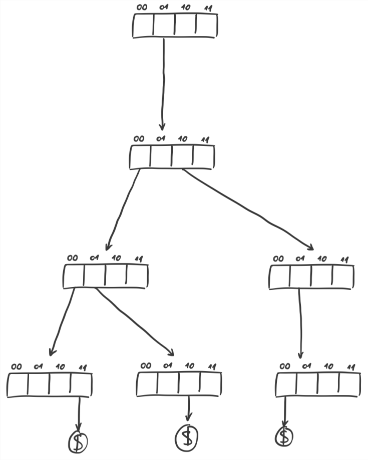
- avoids redundancy
- groups consecutive necklaces

Under the hood:

- store the prefixes in a dynamic bitvector supporting rank/select [Marchini & Vigna 20, Pibiri & Kanda 21]
- associate suffix buckets using a tiered vector (for fast dynamic insertions) [Bille et al. 17]

The structure of the buckets changes dynamically as we add/remove k-mers

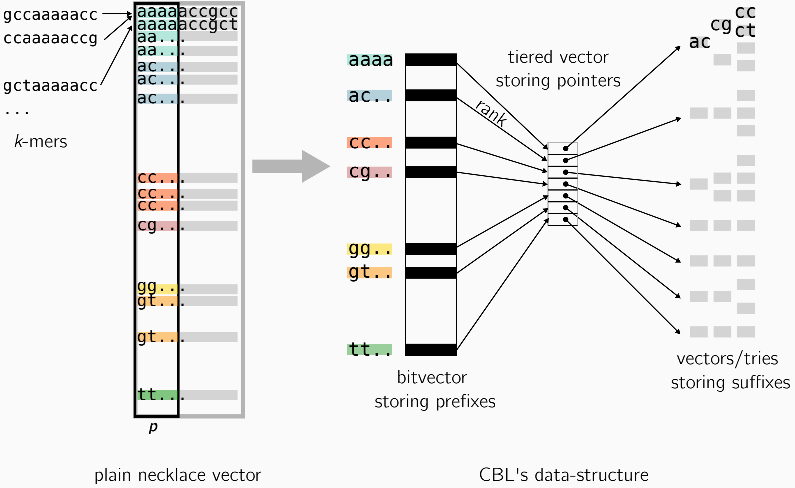
- for small buckets: packed vector, linear search
- for large buckets: trie, logarithmic search



Layout of CBL's data structure

1. compute $\langle x \rangle$
2. split $\langle x \rangle$ as $q \mid\mid r$
3. query r in the bucket of q

$\rightarrow$ faster for consecutive k-mers

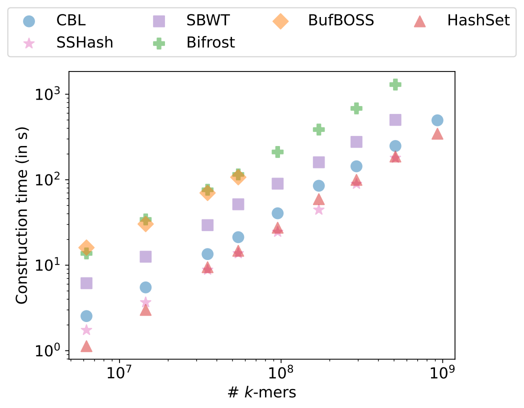


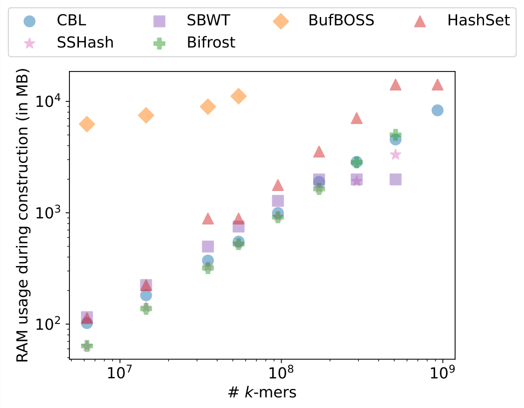
Comparison to some existing tools

category	data structure	membership	insert	delete	∪ ∩ \
BWT	FM-index	✓	✗	✗	✗
—	SBWT	✓	✗	✗	✗
—	dynamic BOSS	✓	✓	✓	✗
hashing	SSHash	✓	✗	✗	✗
—	Bifrost	✓	✓	✗	✗
—	Brisk	✓	✓	✗	✗
—	Bloom filter	approx	✓	✗	union
—	Quotient filter	approx*	✓	✗	union
other	Conway-Bromage	✓	✓	✓	✓
—	**CBL**	✓	✓	✓	✓

*exact if a PHF is used

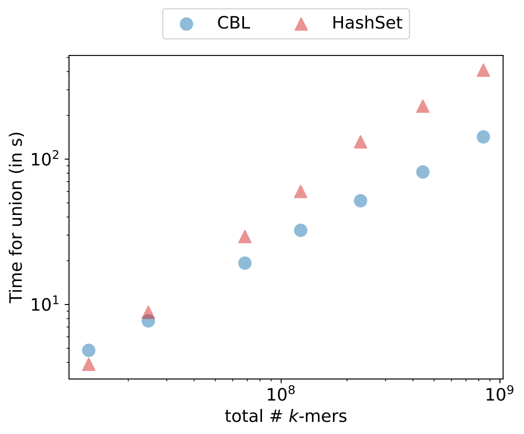
Time/memory usage for collections of bacterial genomes from RefSeq

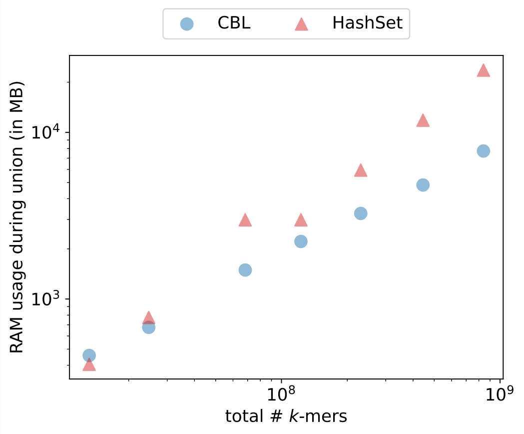




TLDR: almost as fast as a hash table, more memory-efficient

Merging collections of bacterial genomes from RefSeq





TLDR: 4× faster and 3× smaller than a hash table when merging a billion 31-mers

What's next?

Future steps

Improvements of the current structure:

- handle streams of k-mers
- improve buckets' memory usage
  (some ideas: smaller single buckets, adaptive radix trie)
- use SIMD for core operations

Extending the structure:

- concurrent version of CBL
  (distribute suffix buckets between threads)
- associate data (e.g. count) to each k-mer ($\rightarrow$ map structure)

[Your suggestion here]: let's discuss!

Take-home messages

- new dynamic structure based on necklaces
- available as a CLI and a Rust library
- very fast queries, cache efficient
- limited memory usage
- scales for large collections
- versatile operations

Thank you!

Try it here:

`github.com/imartayan/CBL`

Preprint
(accepted to ISMB)

References

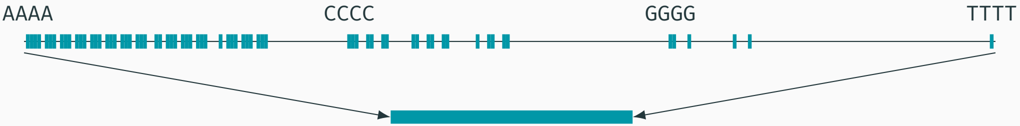
- Alanko, Jarno N, Simon J Puglisi & Jaakko Vuohtoniemi (2022). **“Succinct k-mer sets using subset rank queries on the spectral burrows-wheeler transform”.** In: *bioRxiv*, pp. 2022–05.
- Bille, Philip et al. (2017). **“Fast dynamic arrays”.** In: *arXiv preprint arXiv:1711.00275*.
- Conway, Thomas C & Andrew J Bromage (2011). **“Succinct data structures for assembling large genomes”.** In: *Bioinformatics* 27.4, pp. 479–486.
- Marchini, Stefano & Sebastiano Vigna (2020). **“Compact Fenwick trees for dynamic ranking and selection”.** In: *Software: Practice and Experience* 50.7, pp. 1184–1202.
- Pibiri, Giulio Ermanno & Shunsuke Kanda (2021). **“Rank/select queries over mutable bitmaps”.** In: *Information Systems* 99, p. 101756.
- 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.

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



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