Conway-Bromage-Lyndon (CBL):

an exact, dynamic representation of k-mer sets

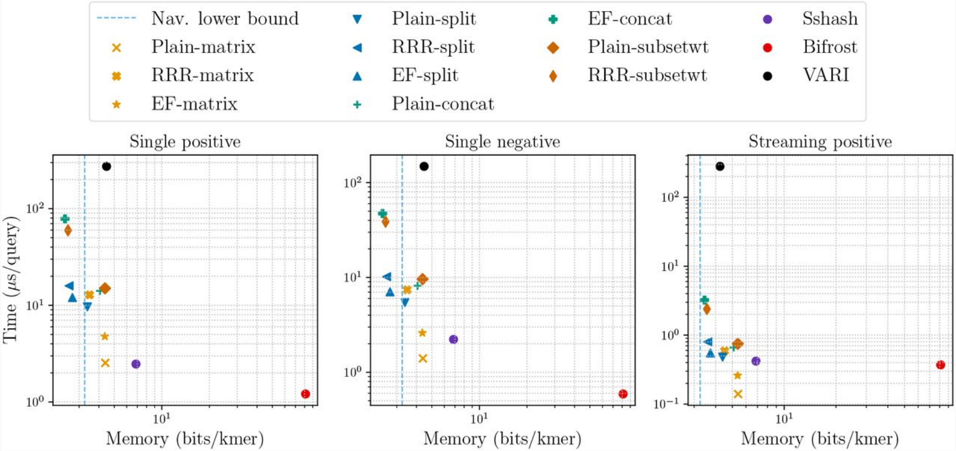
Igor Martayan, Bastien Cazaux, Antoine Limasset & Camille Marchet
University of Lille

July 14, 2024

ISMB 2024 — Montreal

Motivation of this work

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

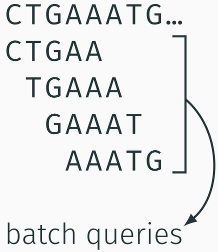


Query time and memory usage of some efficient data structures, taken from [Alanko et al. 23]

Our focus for this talk

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

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



Use case: build an index incrementally, merge/intersect multiple indexes...

Necklace transformation of k-mers

necklace:
smallest cyclic rotation of a word

```
CGAACT

CGAACT (0)
GAACTC (1)
AACTCG (2)
ACTCGA (3)
CTCGAA (4)
TCGAAC (5)
```

$x \longmapsto (\langle x \rangle$, rotation index) is reversible

Amortized necklace computation

Consecutive necklaces can be computed in $\mathcal{O}(\log k)$ amortized time

In practice: $\sim$ 10 ns / necklace

Locality of the necklace transformation

k-mer view

GTCGTTCTTCCTAACGTCATCTCTCATTCTG
TCGTTCTTCCTAACGTCATCTCTCATTCTGT
CGTTCTTCCTAACGTCATCTCTCATTCTGTG
GTTCTTCCTAACGTCATCTCTCATTCTGTGA
TTCTTCCTAACGTCATCTCTCATTCTGTGAC
TCTTCCTAACGTCATCTCTCATTCTGTGACA
CTTCCTAACGTCATCTCTCATTCTGTGACAC
TTCCTAACGTCATCTCTCATTCTGTGACACG
TCCTAACGTCATCTCTCATTCTGTGACACGC
CCTAACGTCATCTCTCATTCTGTGACACGCA
CTAACGTCATCTCTCATTCTGTGACACGCAG
TAACGTCATCTCTCATTCTGTGACACGCAGG
AACGTCATCTCTCATTCTGTGACACGCAGGG
ACGTCATCTCTCATTCTGTGACACGCAGGGT

Locality of the necklace transformation

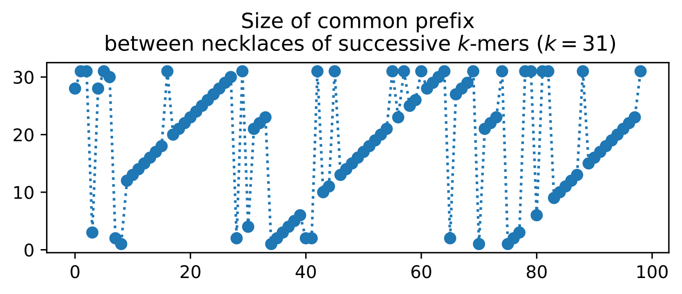
necklace view	k-mer view
AACGTCATCTCTCATTCTG GTCGTTCTTCCT	GTCGTTCTTCCTAACGTCATCTCTCATTCTG
AACGTCATCTCTCATTCTGT TCGTTCTTCCT	TCGTTCTTCCTAACGTCATCTCTCATTCTGT
AACGTCATCTCTCATTCTGTG CGTTCTTCCT	CGTTCTTCCTAACGTCATCTCTCATTCTGTG
AACGTCATCTCTCATTCTGTGA GTTCTTCCT	GTTCTTCCTAACGTCATCTCTCATTCTGTGA
AACGTCATCTCTCATTCTGTGAC 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	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

Locality of the necklace transformation

necklace view

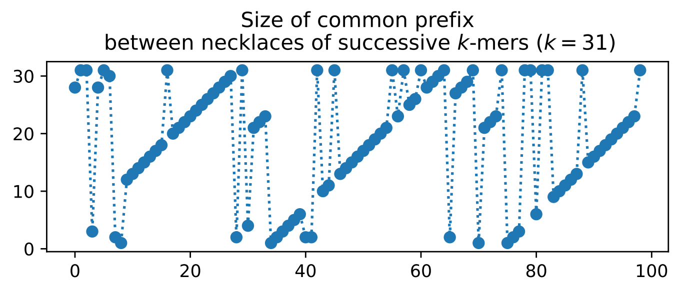
```
AACGTCATCTCTCATTCTG GTCGTTCTTCCT
AACGTCATCTCTCATTCTGT TCGTTCTTCCT
AACGTCATCTCTCATTCTGTG CGTTCTTCCT
AACGTCATCTCTCATTCTGTGA GTTCTTCCT
AACGTCATCTCTCATTCTGTGAC TTCTTCCT
AACGTCATCTCTCATTCTGTGACA TCTTCCT
AACGTCATCTCTCATTCTGTGACAC CTTCCT
AACGTCATCTCTCATTCTGTGACACG TTCCT
AACGTCATCTCTCATTCTGTGACACGC TCCT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



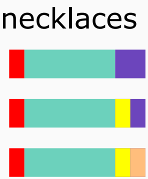
Locality of the necklace transformation

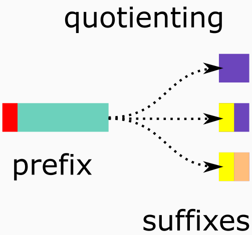
necklace view

```
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
```

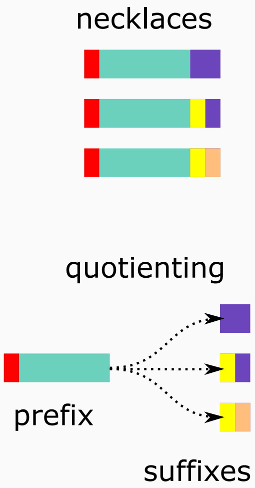


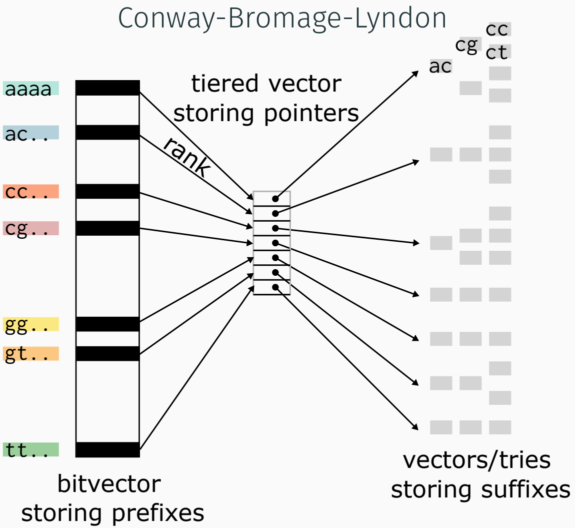
How to exploit these common prefixes?


necklaces


quotienting
prefix
suffixes

Designing a data structure to store necklaces (CBL)



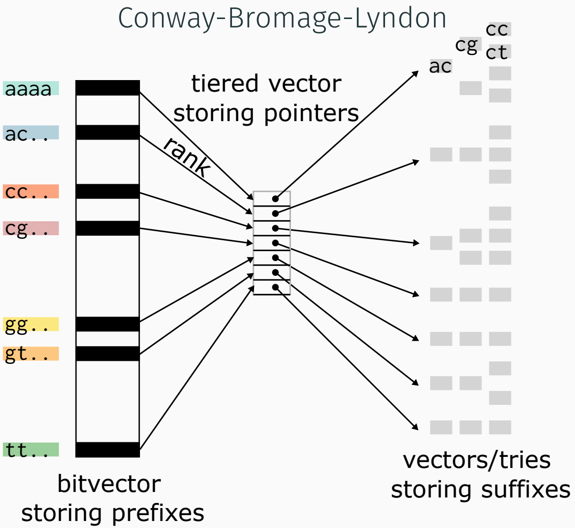


Designing a data structure to store necklaces (CBL)

Main query steps:

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

→ faster for consecutive k-mers
(likely in the same bucket)

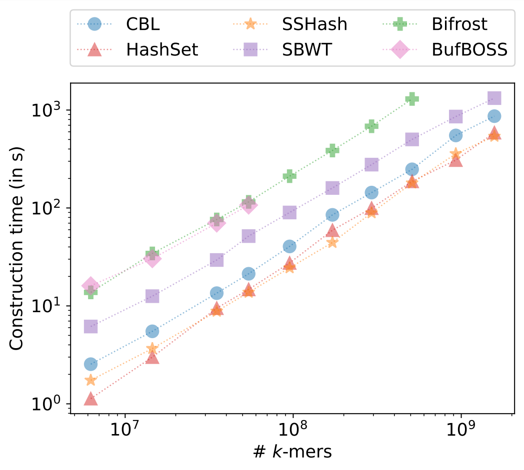


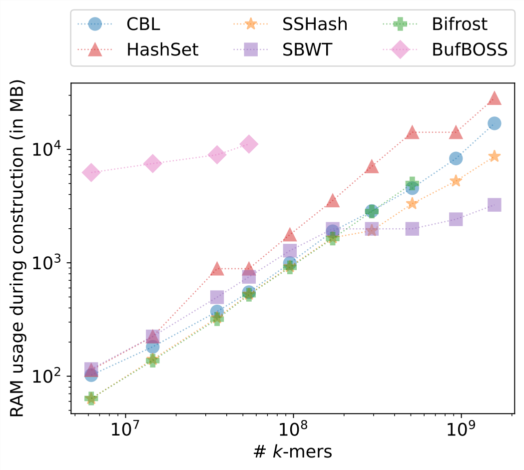
Comparison with some k-mer set data structures

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

*exact if a perfect hash function is used

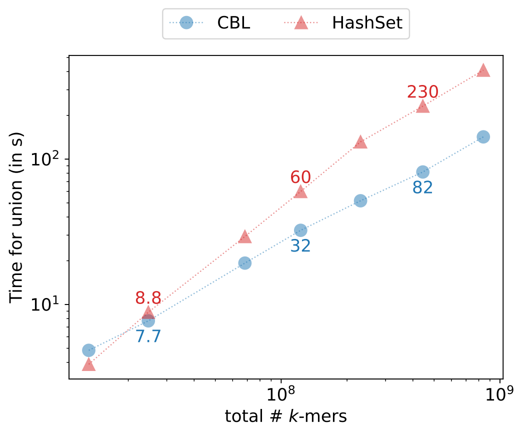
Time/space usage for collections of bacterial genomes from RefSeq ($k = 31$)

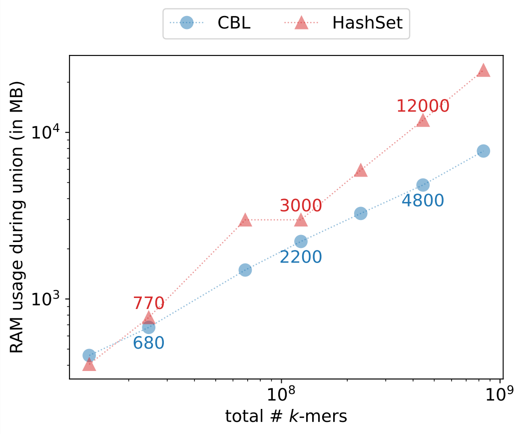




TLDR: almost as fast as a hash table, ∼40–50 bits / k-mer ($k = 31$)

Merging collections of bacterial genomes from RefSeq (k = 31)





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

What's next?

Improving CBL's memory usage:

- suffixes among the same bucket are similar and can be compressed
- better layout of the tries (e.g. adaptive radix tries)

Extending the data structure:

- associate data (e.g. abundance) to each k-mer $\rightarrow$ CBL Map
- concurrent version (distribute suffix buckets between threads)

Using CBL to enumerate k-mers satisfying a given constraint
e.g. find k-mers present in ref A and B but not in C $\longrightarrow$ preprint:

Take-home messages

- new dynamic structure based on necklaces
- very fast queries, cache efficient
- limited memory usage ($\sim$ 40 bpk for k=31)
- supports fast insertion, deletion & set ops
- available as a CLI and a Rust library

github.com/imartayan/CBL

Paper

Thank you!

References

- Alanko, Jarno N., Simon J. Puglisi & Jaakko Vuohtoniemi (2023). **“Small Searchable κ-Spectra via Subset Rank Queries on the Spectral Burrows-Wheeler Transform”.** In: *ACDA23*, pp. 225–236.
- Bille, Philip et al. (2017). **“Fast Dynamic Arrays”.** In: LIPIcs 87, 16:1–16:13. ISSN: 1868-8969.
- 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.

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

Quickly computing streams of necklaces

Faster necklace computation

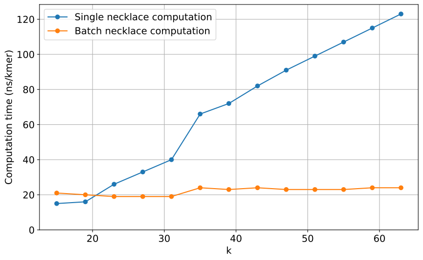
Only consider the cyclic rotations that start:

- 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)

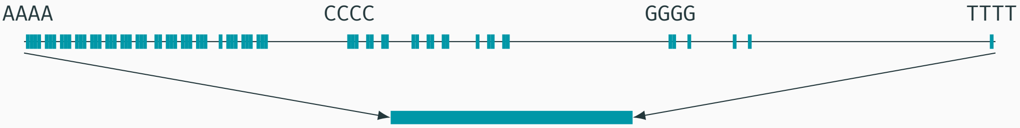


DENSIFIYING THE SPACE OF NECKLACES BY RANKING

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