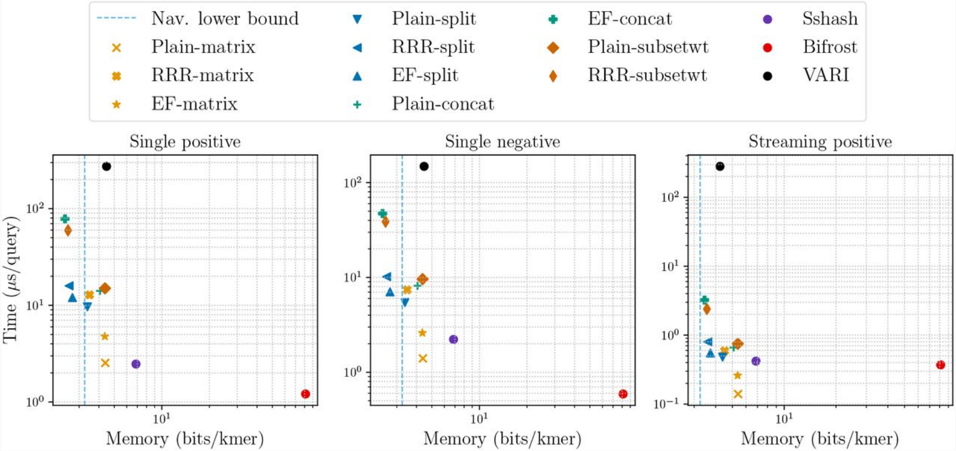
A space-efficient, locality-preserving and dynamic data structure for indexing k-mers

Igor Martayan, Bastien Cazaux, Antoine Limasset & Camille Marchet

November 21, 2023

SeqBIM 2023 — Lille

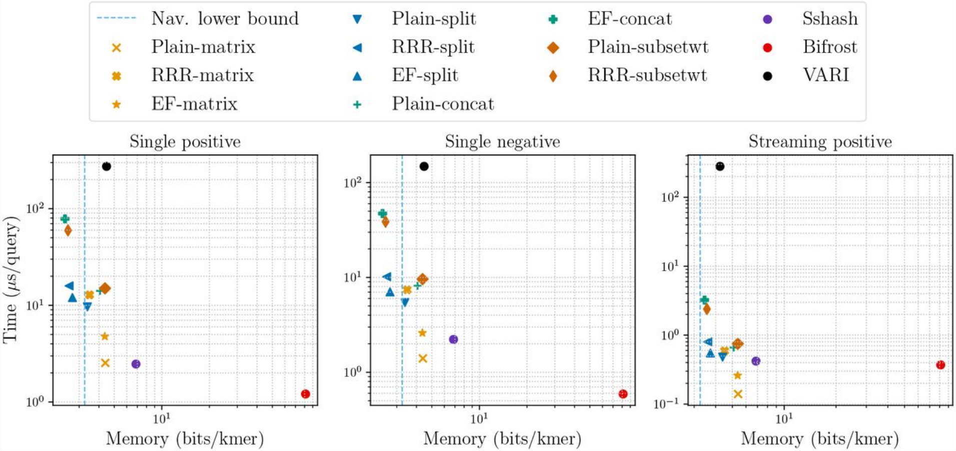
Plenty of compact data structures for storing k-mers



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

Motivation

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 $4^k/3$

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 $4^k/3$

How can we improve this approach?

Wish list for an ideal data structure

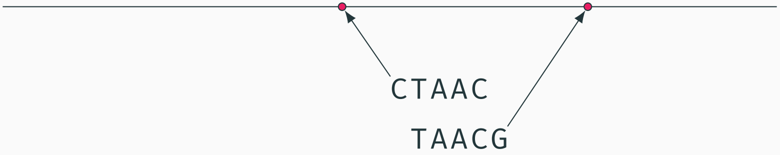
- space-efficient: few bits / k-mer
- dynamic: support insertion and deletion after construction
- efficient queries:
  - membership
  - enumeration
  - insertion
  - (deletion)
- locality-preserving: reduce cache misses when querying consecutive k-mers

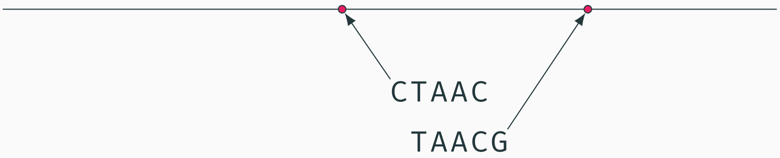
```
CTGAAATG…
CTGAA
 TGAAA
  GAAAT
   AAATG
```

batch queries

Preserving k-mer locality

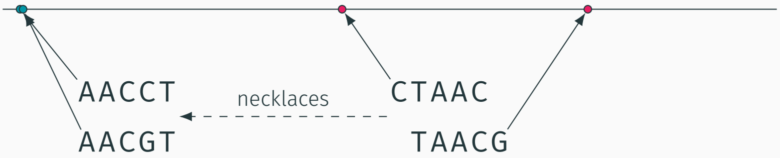
A LOCALITY-PRESERVING ENCODING OF K-MERS





Alternative encoding based on necklaces

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



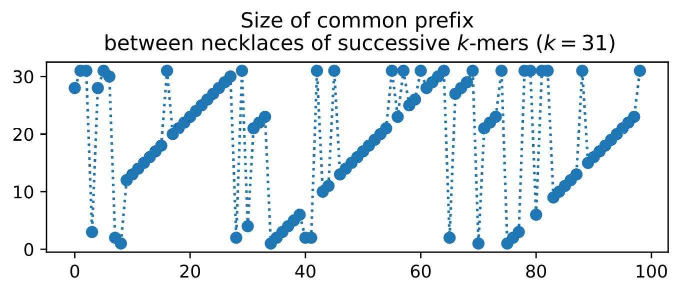
Alternative encoding based on necklaces

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 bijective 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



Ranking necklaces to improve compression

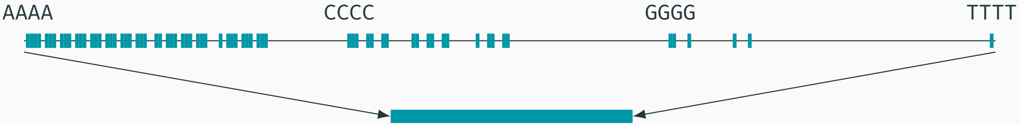
The number of necklaces of size k on an alphabet with σ letters is $\sim \dfrac{\sigma^k}{k}$

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

Ranking necklaces to improve compression

The number of necklaces of size k on an alphabet with σ letters is $\sim \dfrac{\sigma^k}{k}$

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]

(Can we do better? for batch queries maybe?)

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

Compressing sparse integer sets

Compressing sparse integer sets with Elias-Fano encoding

[Elias 74, Fano 71]

- separate the high bits and low bits
- compress them with different methods

We choose the size of the low bits as $l = \left\lceil \lg \frac{u}{n} \right\rceil$

- n is the number of elements
- u is the size of the universe
  e.g. $u = 4^k$ for k-mers

$\{2, 3, 210, 216, 231, 265, 491, 499\}$

	000	000010
	000	000011
	011	010010
slowly	011	011000
increasing	011	100111
	100	001001
	111	101011
	111	110011
		$\xleftrightarrow{\quad l \quad}$

Almost optimal space usage

Space usage of Elias-Fano

$$EF(n,u) = 2n + n\left\lceil \lg \frac{u}{n} \right\rceil$$

e.g. for $n = 10^{10}$ and $u = 4^{31}$, EF uses 31 bits / item

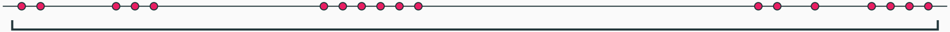
Information theoretic lower bound

$$\begin{aligned}\lg \binom{u}{n} &\approx n \lg e + n \lg \frac{u}{n} \\ &\approx 1.44n + n \lg \frac{u}{n}\end{aligned}$$

Note that the bound can get lower if we have additional knowledge about the distribution.

Partitioning sparse integer sets

Partitioning sparse integer sets [Ottaviano & Venturini 14]



lot of empty regions

Split the sequence into smaller blocks

Partitioning sparse integer sets [Ottaviano & Venturini 14]

Split the sequence into smaller blocks, choose the best encoding:

- for sparse blocks: Elias-Fano ; $2n + n\lceil \lg \frac{u}{n} \rceil$ bits
- for dense blocks: plain bitset ; u bits
- for full blocks: lower bound + size is enough

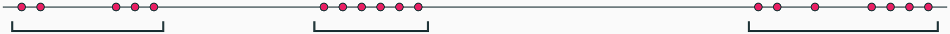
Partitioning sparse integer sets [Ottaviano & Venturini 14]

Split the sequence into smaller blocks, choose the best encoding:

- for sparse blocks: Elias-Fano ; $2n + n\left\lceil \lg \frac{u}{n} \right\rceil$ bits
- for dense blocks: plain bitset ; u bits
- for full blocks: lower bound + size is enough

Computing the optimal partition

- optimal solution in $\mathcal{O}\left(n^2\right)$ using dynamic programming
- $(1+\varepsilon)$-approximation in $\mathcal{O}\left(n \cdot \frac{1}{\varepsilon} \ln \frac{1}{\varepsilon}\right)$



[Pibiri & Venturini 17] presents an approach to make the partitions dynamic using $o(n)$ extra space

$\rightarrow$ WIP, *no practical implementation available yet*

Query complexity

- membership and successor in $\mathcal{O}(\lg\lg n)$
- insertion and deletion in $\mathcal{O}(\lg n \,/\, \lg\lg n)$

[Pibiri & Venturini 17] presents an approach to make the partitions dynamic using $o(n)$ extra space

$\rightarrow$ WIP, *no practical implementation available yet*

Query complexity

- membership and successor in $\mathcal{O}(\lg \lg n)$
- insertion and deletion in $\mathcal{O}(\lg n \,/\, \lg \lg n)$

- ranking saves $\lg k$ bits / k-mer but costs $\mathcal{O}(k^2)$ / query
- partitioning typically saves $\frac{1}{2}\lg k$ bits / k-mer

Conclusion

Take-home messages

Using necklaces to represent k-mers

- preserves locality
- improves compression

Partitioned sparse sets

- fit in well with necklace locality
- can support dynamic operations

Future steps

- efficient implementation of the dynamic partitions
- batch necklace computation
- batch rank computation
- subquadratic ranking?
- bound on the necklace distance

Take-home messages

Using necklaces to represent k-mers

- preserves locality
- improves compression

Partitioned sparse sets

- fit in well with necklace locality
- can support dynamic operations

Future steps

- efficient implementation of the dynamic partitions
- batch necklace computation
- batch rank computation
- subquadratic ranking?
- bound on the necklace distance

Thank you!

Appendix

- 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.
- Elias, Peter (1974). **"Efficient storage and retrieval by content and address of static files".** In: *Journal of the ACM (JACM)* 21.2, pp. 246–260.
- Fano, Robert Mario (1971). ***On the number of bits required to implement an associative memory.*** Massachusetts Institute of Technology, Project MAC.
- Ferragina, Paolo, Igor Nitto & Rossano Venturini (2011). **"On optimally partitioning a text to improve its compression".** In: *Algorithmica* 61, pp. 51–74.

References II

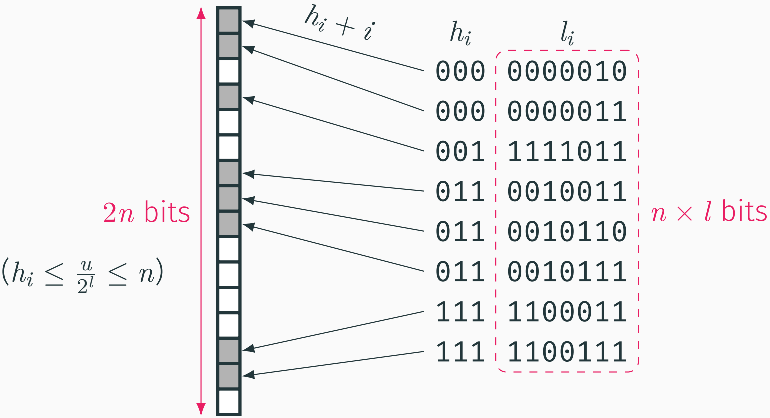
Ottaviano, Giuseppe & Rossano Venturini (2014). **“Partitioned elias-fano indexes”.** In: *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pp. 273–282.

Pibiri, Giulio Ermanno & Rossano Venturini (2017). **“Dynamic elias-fano representation”.** In: *28th Annual symposium on combinatorial pattern matching (CPM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

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.

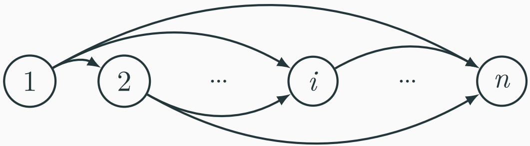
A closer look at Elias-Fano encoding [Elias 74, Fano 71]

$S = \{2, 3, 251, 403, 406, 407, 995, 999\}$ $\quad n = 8 \quad u = 1000 \quad l = \lceil \lg \frac{u}{n} \rceil = 7$ bits



Optimal partition as a shortest path [Ferragina et al. 11]

- $V = [\![1, n]\!] \quad E = \{i < j \; ; i, j \in V\}$
- $w_{i,j}$ = cost to encode $S[i, j]$



Computing the optimal partition

- optimal solution in $\mathcal{O}(|V| + |E|) = \mathcal{O}(n^2)$ using dynamic programming
- $(1+\varepsilon)$-approximation in $\mathcal{O}(n \cdot \frac{1}{\varepsilon} \ln \frac{1}{\varepsilon})$ by sparsifying the graph