The anatomy of an order-preserving k-mer dictionary

Giulio Ermanno Pibiri

Ca' Foscari University of Venice

@giulio_pibiri

@jermp

RIKEN-AIP Tutorial: Compressed Data Structures for Advanced Data Analysis
Tokyo, Japan, 2 July 2025

Agenda

1. Context, motivations, and problem definition
2. Tools:
    - Spectrum-preserving string sets
    - Minimizers
    - Minimal perfect hashing
3. Sparse and skew hashing of k-mers

1. Context, motivations, and problem definition

Massive DNA Collections

ENA
European Nucleotide Archive

e!Ensembl

NIH National Library of Medicine
National Center for Biotechnology Information

- **Peta bytes** of data available:
  - ENA (European Nucleotide Archive)
  - SRA (Sequence Read Archive)
  - RefSeq (Reference Sequence Database)
  - Ensembl

- For example: as of June 2025, ENA has 5.7 billions of assembled sequences, for 30 trillion bases.
  https://www.ebi.ac.uk/ena/browser/about/statistics

- These collections are paving the way to answer fundamental questions regarding biology and evolution.

k-mers

- **Q.** But how do we exploit such potential?

  We need efficient methods to index and search data at this scale.

- One popular strategy: “reduce” a DNA sequence to a set of short substrings of fixed length k — the so-called ***k-mers***.

```
ACGGTAGAACCGATTCAAATTCGACGTAGC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     AGAACCGATTCAA
      GAACCGATTCAAA
       AACCGATTCAAAT
         ...
```

← Example for k=13.

k-mer applications

- Software tools based on k-mers are predominant in bioinformatics.
- Many applications:
  - genome assembly
  - variant calling
  - pan-genome analysis
  - meta-genomics
  - sequence comparison/alignment

  …

A world of k-mer indexes

- Huge research effort produced many types of indexes based on k-mers, with different:

  - representations (hashing, BWT-based, exact vs. approximate),
  - features (e.g., static vs. dynamic),
  - space/time trade-offs,
  - types of supported queries, etc.

- Recent surveys on this topic:

  - Data Structures based on *k*-mers for Querying Large Collections of Sequencing Data Sets
    Marchet et al., Genome Research, 2020.

  - Data Structures to Represent a Set of *k*-long DNA Sequences
    Chikhi et al., ACM Computing Surveys, 2021.

The k-mer dictionary problem

- We are given a large DNA string S (e.g., a genome or a pan-genome) and let K be the set of all its n distinct k-mers.

  Example: The human genome (GRCh38) has >2.5B distinct k-mers for k=31.

- **Problem.** We want to build a dictionary for K so that the following operations are efficient:
  - Lookup(x) returns $1 \leq i \leq n$ if the k-mer $x \in K$ or $\perp$ otherwise;
  - return the k-mer $x =$ Access(i) if $1 \leq i \leq n$.

- Other operations of interest are **streaming queries**, iteration, navigational queries.

Do we need an *ad-hoc* solution?

- The algorithmic literature about *(compressed) string dictionaries* is rich of solutions [Martínez-Prieto et al., 2016] (e.g., Front-Coding, path-decomposed tries, double-array tries, etc.).

- But they are relevant for “generic strings”:
  - variable-length,
  - larger alphabets (e.g., ASCII),
  - (usually) no particular properties of the strings to aid compression.

- Since k-mers are extracted *consecutively* from DNA, a k-mer following another one shares k-1 bases (very low entropy).

```
ACGGTAGAACCGATTCAAATTCGACGTAGC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     ...
```

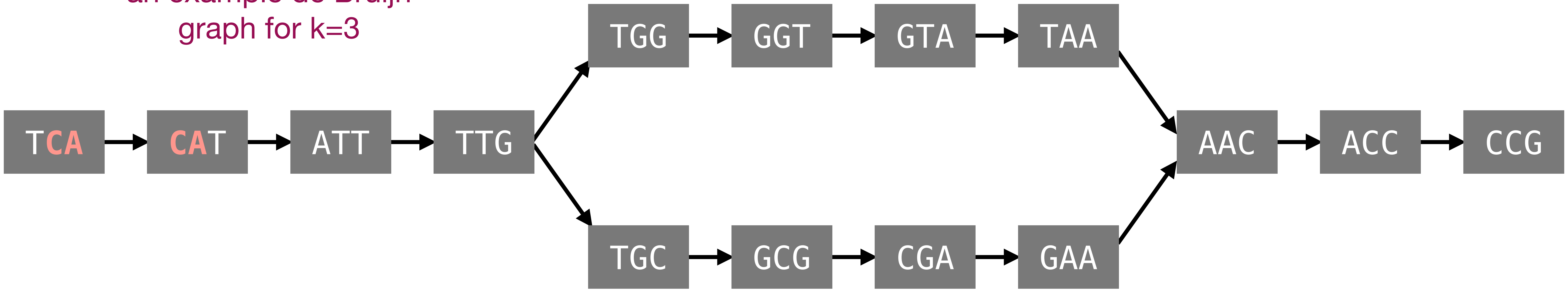
← Example for k=13.

2. Tools

de Bruijn graphs

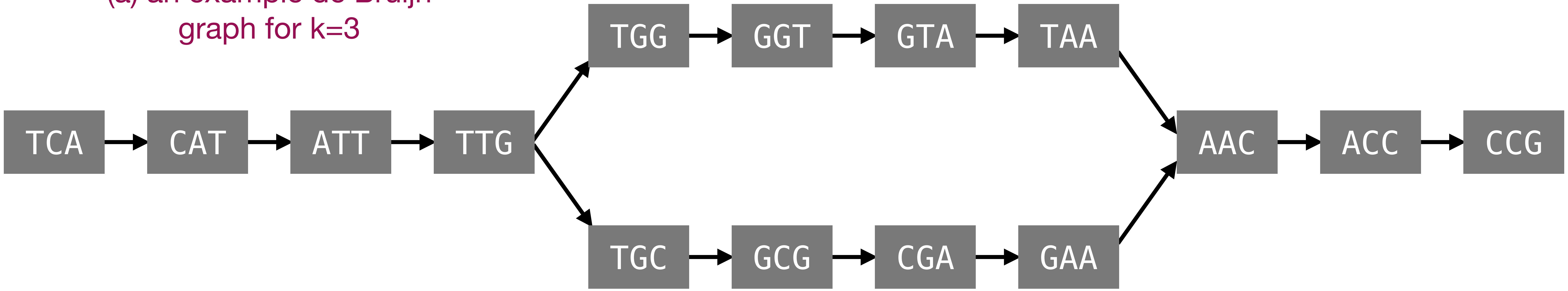
- **de Bruijn graph.** A (node centric) de Bruijn graph (dBG) of order k for a sequence S is a directed graph where nodes are the distinct k-mers of S and there exists a directed edge from x to y if $x[2..k] = y[1..k-1]$.
- **Fact.** Equivalence between a set of k-mers and a de Bruijn graph.

an example de Bruijn graph for k=3

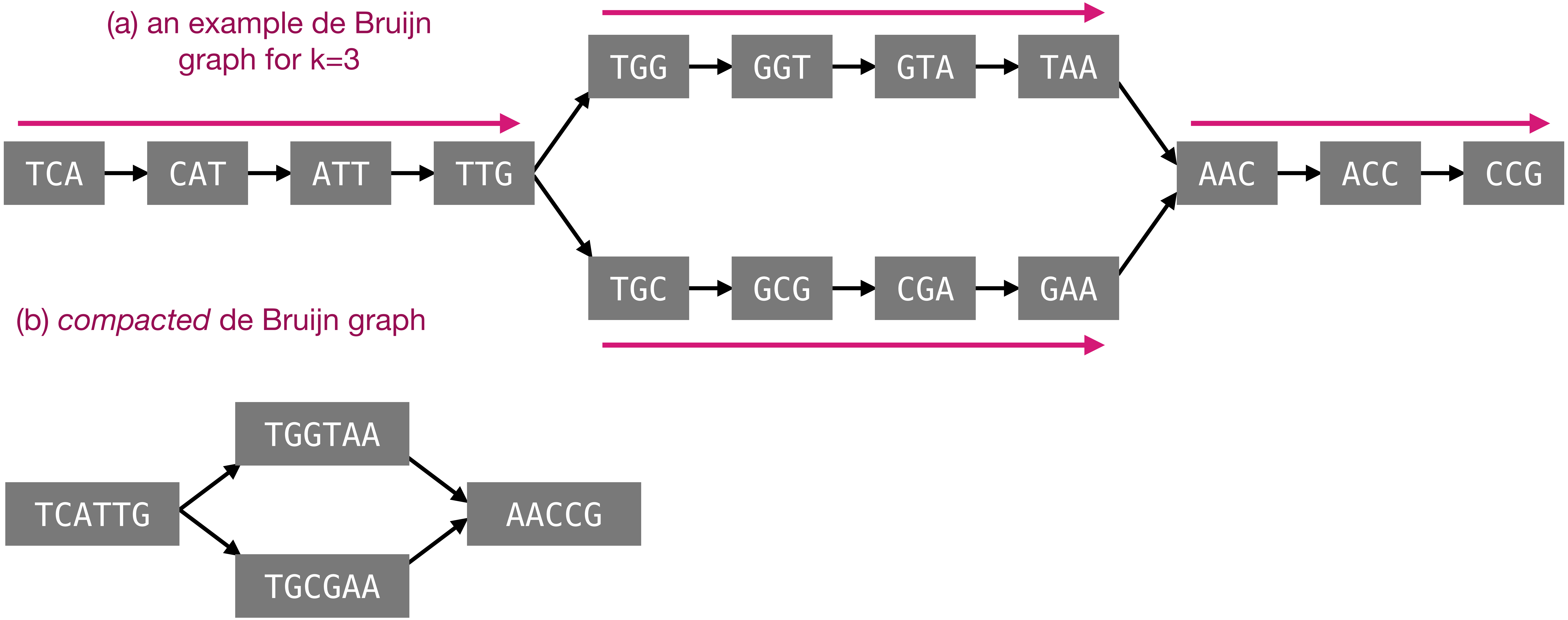


de Bruijn graphs

(a) an example de Bruijn graph for k=3



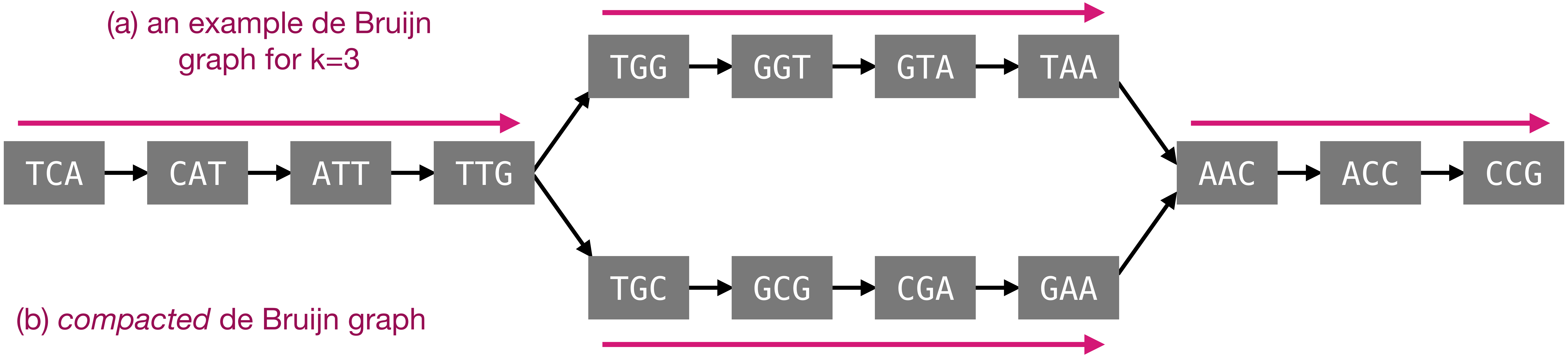
de Bruijn graphs


(a) an example de Bruijn graph for k=3

(b) *compacted* de Bruijn graph

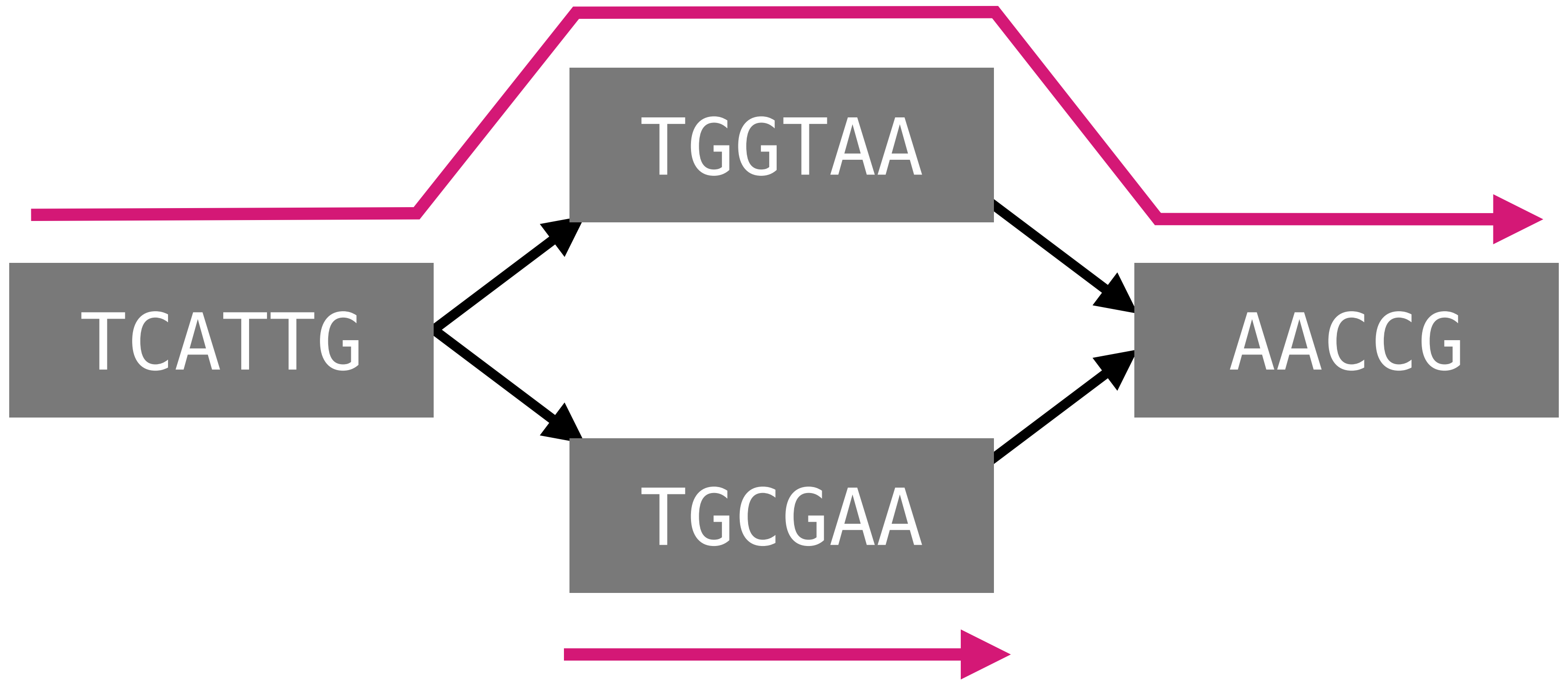
de Bruijn graphs

(a) an example de Bruijn graph for k=3



TCA
CAT
ATT
TTG
TGG
GGT
GTA
TAA
TGC
GCG
CGA
GAA
AAC
ACC
CCG

(b) *compacted* de Bruijn graph



TCATTG
TGGTAA
TGCGAA
AACCG

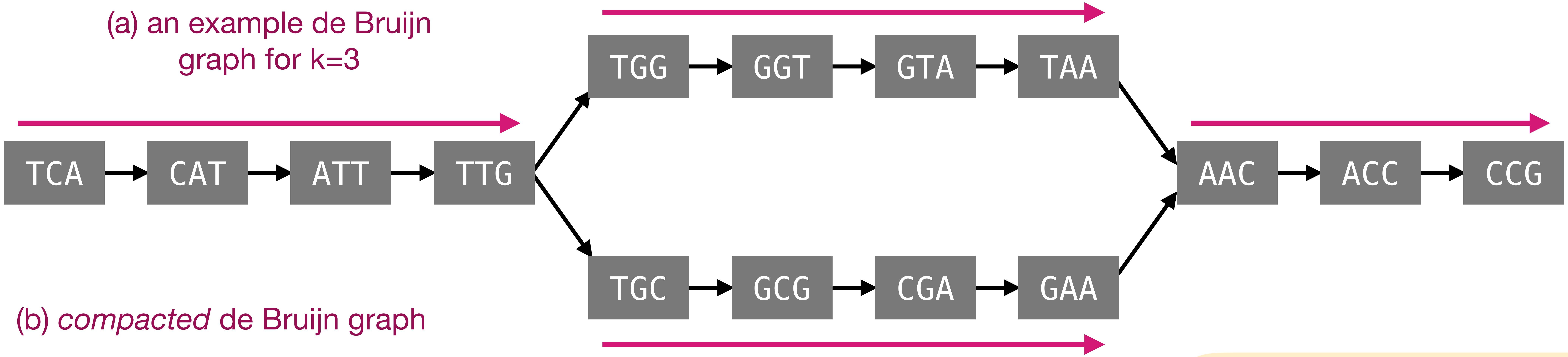
(c) set of “stitched” unitigs

TCATTGGTAACCG

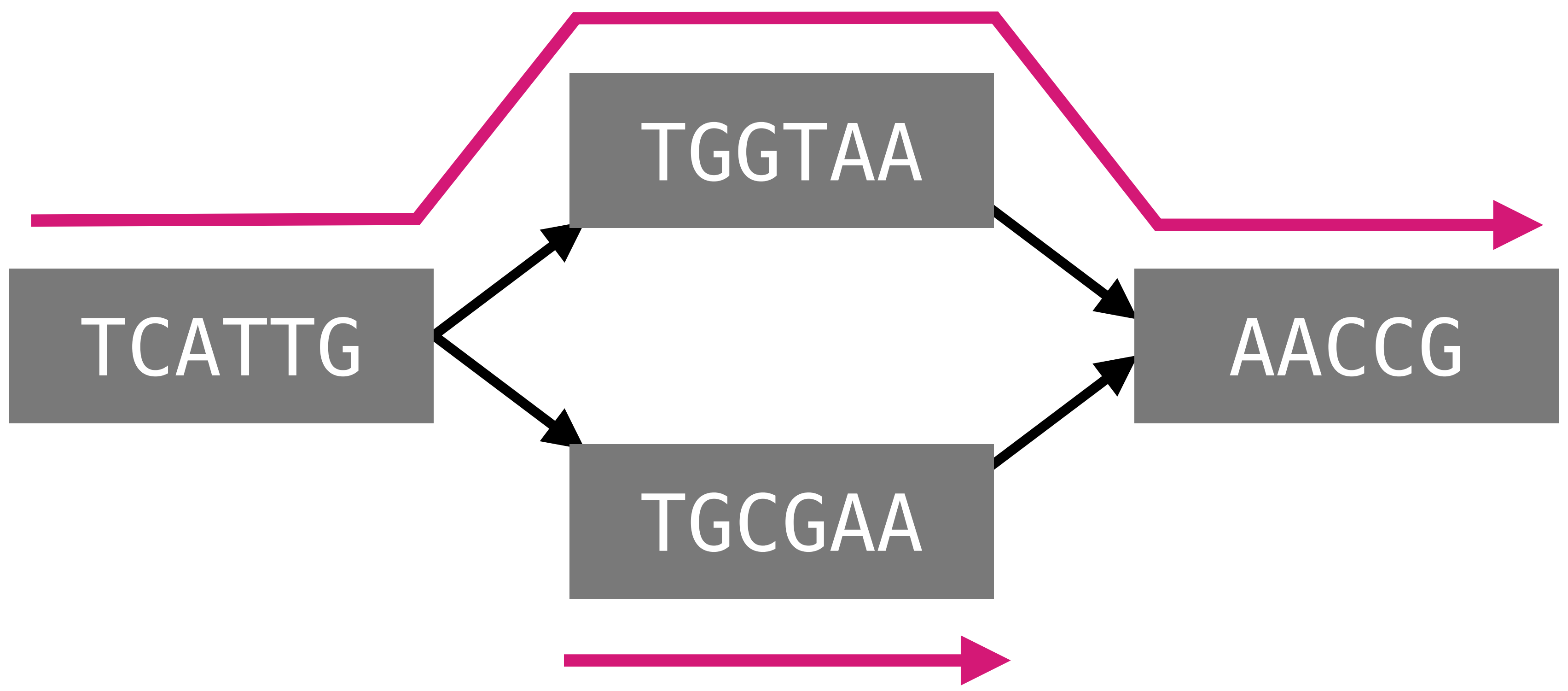
TGCGAA

de Bruijn graphs

(a) an example de Bruijn
graph for k=3



(b) *compacted* de Bruijn graph



(c) set of “stitched” unitigs

TCATTGGTAACCG

TGCGAA

this is an example of a
spectrum-preserving string set
(see next)

Spectrum-preserving string sets

- **k-mer spectrum.** The spectrum of S, say $spect(S)$, is the set of all the distinct k-mers of S.
- **Spectrum-preserving string set (SPSS).** A SPSS for S is a set of strings $\mathcal{U} = \{U_1, \ldots, U_m\}$ such that $spect(S) = spect(U_1) \cup \cdots \cup spect(U_m)$. Usually $spect(U_i) \cap spect(U_j) = \emptyset$ for any $i \neq j$.
- For example, the set of unitigs or “stitched” unitigs of the dBG of S are possible SPSS for S.

Spectrum-preserving string sets

- **k-mer spectrum.** The spectrum of S, say $spect(S)$, is the set of all the distinct k-mers of S.
- **Spectrum-preserving string set (SPSS).** A SPSS for S is a set of strings $\mathscr{U} = \{U_1, \ldots, U_m\}$ such that $spect(S) = spect(U_1) \cup \cdots \cup spect(U_m)$. Usually $spect(U_i) \cap spect(U_j) = \emptyset$ for any $i \neq j$.
- For example, the set of unitigs or "stitched" unitigs of the dBG of S are possible SPSS for S.
- In general, we want to **minimise** the cumulative length of $\mathscr{U}$, i.e., the number of characters in the strings of $\mathscr{U}$.
- A **general framework**: compute a minimum-size **path cover** for the (compacted) dBG of S. Usually the cover is **disjoint-node**, so that each k-mer of S appears exactly once in $\mathscr{U}$.

Spectrum-preserving string sets

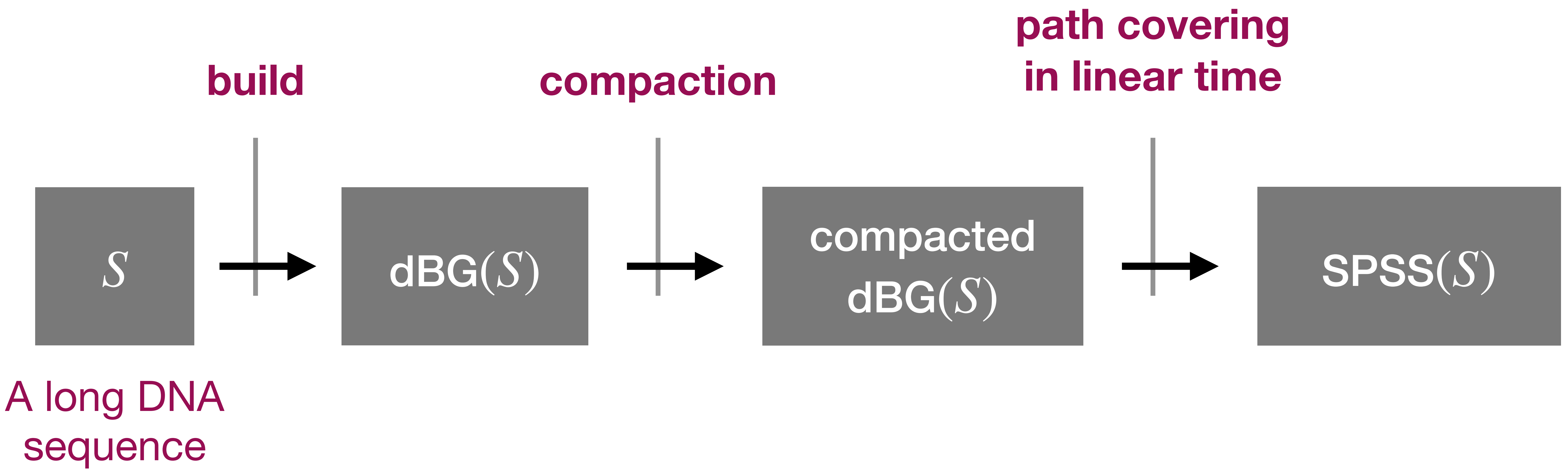
- Many SPSS available in the literature:

  - unitigs (folklore);
  - stitched unitigs [Rahman and Medvedev, 2020] (almost optimal);
  - simplitigs [Brinda et al., 2020];
  - eulertigs [Schmidt and Alanko, 2022, 2023] (**optimal**: smallest num. of characters);
  - matchtigs [Schmidt, Khan, Alanko, P., Tomescu, 2023];
  - masked super strings [Brinda et al., 2025].

- Some of them have slightly different properties.

- For example: unitigs, stitched unitigs, simplitigs, and eulertigs **do not allow repetitions** of k-mers, whereas matchtigs and masked super strings do.

de Bruijn graphs and spectrum-preserving string sets



build

compaction

path covering
in linear time

S

dBG(S)

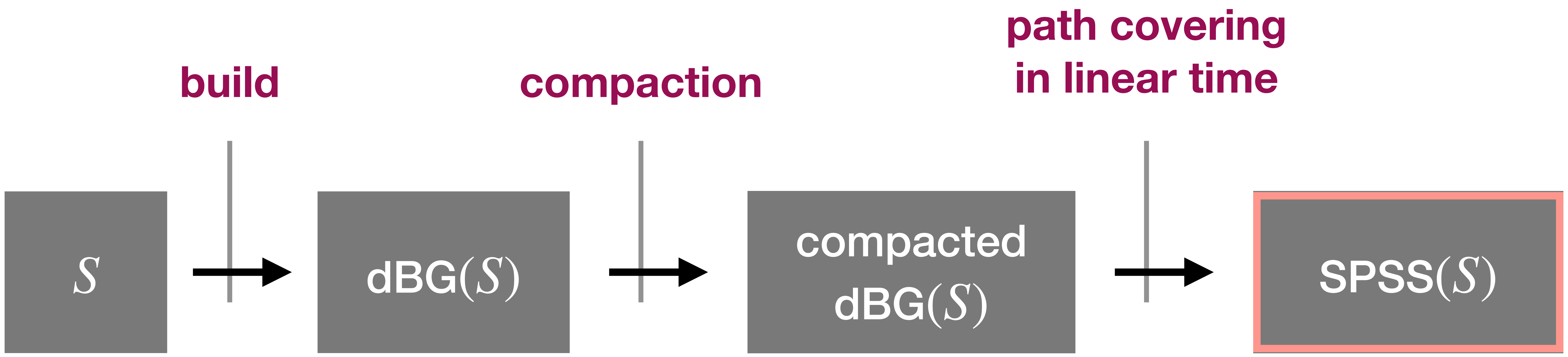
compacted
dBG(S)

SPSS(S)

A long DNA
sequence

- BCALM2 [Chikhi et al., 2016]
- TwoPaCo [Minkin et al., 2017]
- Cuttlefish1 [Khan and Patro, 2021]
- Cuttlefish2 [Khan et al., 2022]
- GGCAT [Cracco and Tomescu, 2023]
- Cuttlefish3 [Khan, Dhulipala and Patro, 2025]

de Bruijn graphs and spectrum-preserving string sets



build

compaction

path covering
in linear time

S

dBG(S)

compacted
dBG(S)

SPSS(S)

A long DNA
sequence

A collection of DNA strings with
no duplicate k-mers:
this is a natural basis for a
space-efficient k-mer dictionary

- BCALM2 [Chikhi et al., 2016]
- TwoPaCo [Minkin et al., 2017]
- Cuttlefish1 [Khan and Patro, 2021]
- Cuttlefish2 [Khan et al., 2022]
- GGCAT [Cracco and Tomescu, 2023]
- Cuttlefish3 [Khan, Dhulipala and Patro, 2025]

Indexing SPSS

- Now that we have an SPSS where each k-mer of S appears once, the question is:

  Q. How do we index it so that Lookup is efficient?

Indexing SPSS

- Now that we have an SPSS where each k-mer of S appears once, the question is:

  Q. How do we index it so that Lookup is efficient?

- Possible answers:

  - Compute the **BWT** of the strings in the SPSS.

  - We are going to see a solution based on **hashing.**
    (We need two more tools.)

Sketching with minimizers

- Consider each k-mer x of S: sample one m-mer of x out of its $k-m+1$ m-mers and call it the “representative” of x — or its *minimizer*.

Example for $k=10$ and $m=7$.

ACGGTAGAACCGATTCAAATTCGAT…

ACGGTAGAAC
CGGTAGAACC
GGT**AGAACCG**
GT**AGAACCG**A
TAG**AACCGAT**
AG**AACCGAT**T
G**AACCGAT**TC
AACCGATTCA
…

Sketching with minimizers

- Consider each k-mer x of S: sample one m-mer of x out of its $k - m + 1$ m-mers and call it the “representative” of x — or its *minimizer*.

Example for $k = 10$ and $m = 7$.

ACGGTAGAACCGATTCAAATTCGAT…

ACGGTAGAAC
CGGTAGAACC
GGT**AGAACCG**
GT**AGAACCG**A
TAG**AACCGAT**
AG**AACCGAT**T
G**AACCGAT**TC
AACCGATTCA
…

- We would like to sample the **same minimizer** from consecutive k-mers so that the **set of distinct minimizers** forms a succinct sketch for S.
- This reduces the memory footprint and computational time of countless applications in Bioinformatics.

Sketching with minimizers

- **Q.** How do we compare different sampling algorithms?

  A. We define the *density* of a sampling algorithm as the fraction between the number of (distinct) minimizers and the total number of m-mers of S (i.e., $|S| - m + 1$).

  The lower the density, the better!

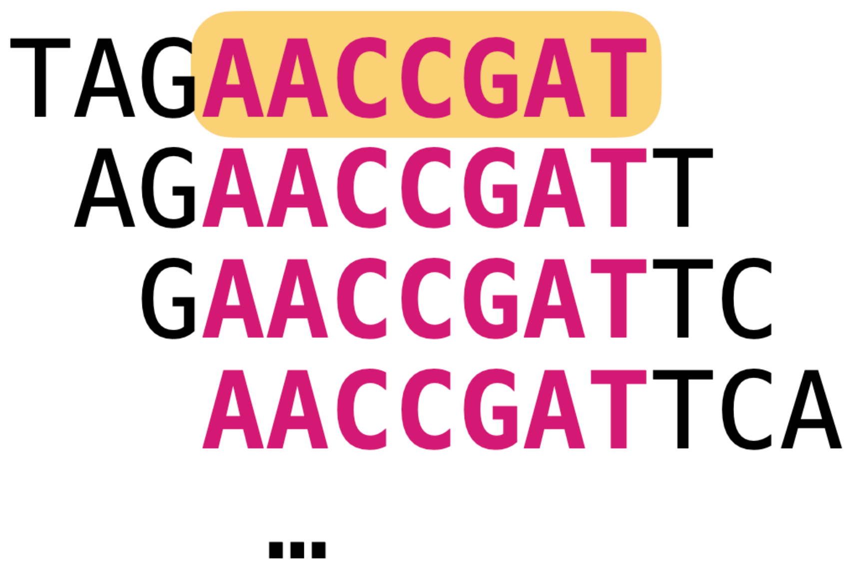
Sketching with minimizers

- **Q.** How do we compare different sampling algorithms?

  A. We define the *density* of a sampling algorithm as the fraction between the number of (distinct) minimizers and the total number of m-mers of S (i.e., $|S| - m + 1$).

  The lower the density, the better!

- Call $w = k - m + 1$. Since the same m-mer cannot be a minimizer for more than w consecutive k-mers, we immediately have a **lower bound** of $1/w$ on the density of any sampling algorithm.



```
TAGAACCGAT
 AGAACCGATT
  GAACCGATTC
   AACCGATTCA
     ...
```

Example for $w = 4$ and $m = 7$.

The “folklore” minimizer

- **Minimizer.** [Schleimer et al. 2003, Roberts et al., 2004] Given a k-mer x and an order $\mathcal{O}$ over all m-mers, the *minimizer* of length $m \leq k$ is the (leftmost) *smallest* m-mer of x according to $\mathcal{O}$.
- Example. Given $x = \mathtt{ACGGTAGAACCGA}$ $(k = 13)$ and $m = 4$:

ACGG
CGGT
GGTA
GTAG
TAGA
AGAA
GAAC
AACC
ACCG
CCGA

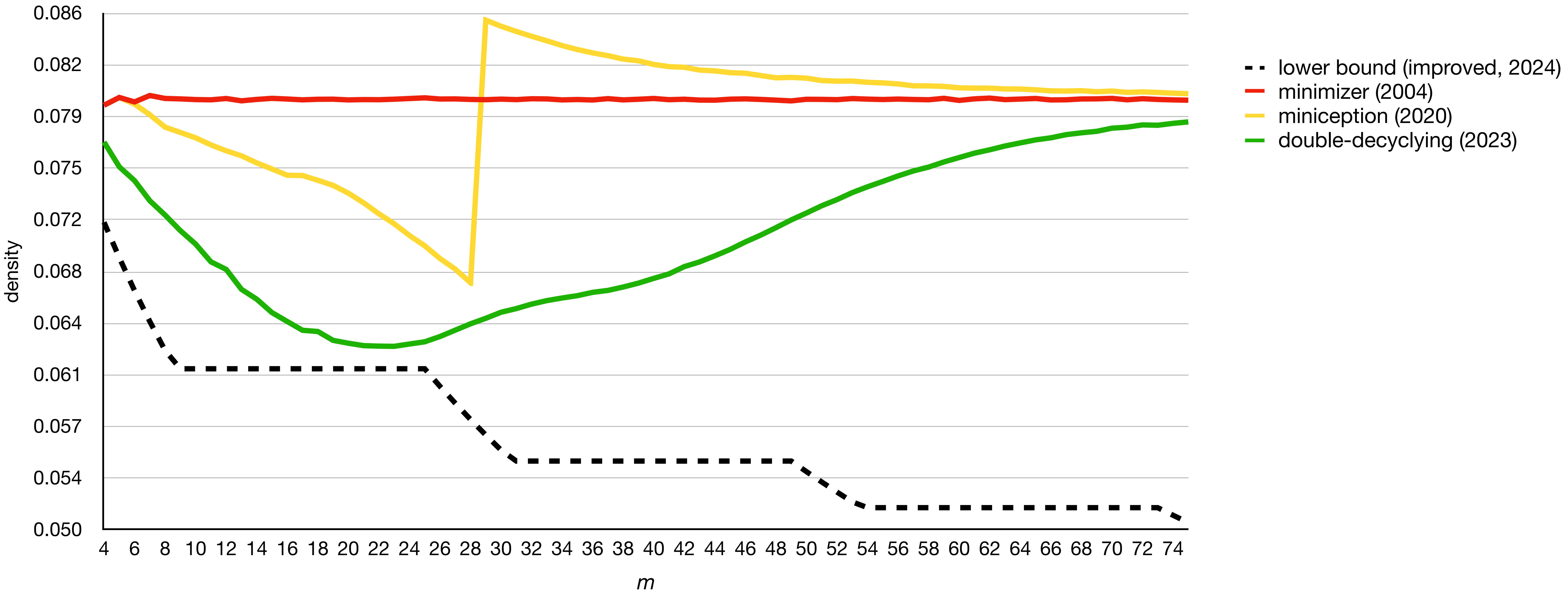
$\mathcal{O}$ is the *lexicographic* order.

$h(\mathtt{ACGG}) = 9842978325$
$h(\mathtt{CGGT}) = 817612312$
$h(\mathbf{GGTA}) = 8265731$ ← *smallest* hash code
$h(\mathtt{GTAG}) = 478491248$
$h(\mathtt{TAGA}) = 17491411$
$h(\mathtt{AGAA}) = 17148914$
$h(\mathtt{GAAC}) = 91815379$
$h(\mathtt{AACC}) = 645793914$
$h(\mathtt{ACCG}) = 918417644$
$h(\mathtt{CCGA}) = 814188124$

$\mathcal{O}$ is defined by a random hash function h.

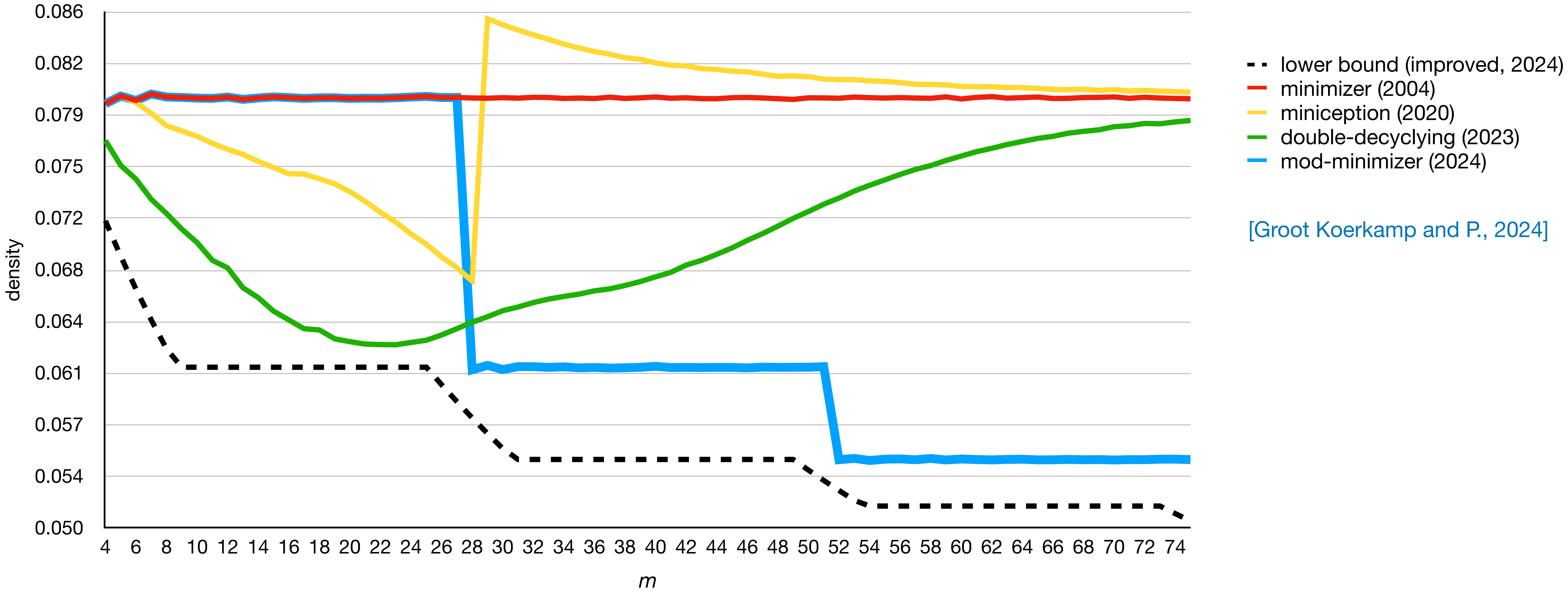
In this case, the density is $2/(w+1)$: almost a factor of 2 away from the lower bound for large w.

Density by varying *m*



- Example for $w = 24$, so $k = 24 + m - 1$.
- Measured over a string of 10 million i.i.d. random characters with an alphabet size of 4.
- https://github.com/jermp/minimizers

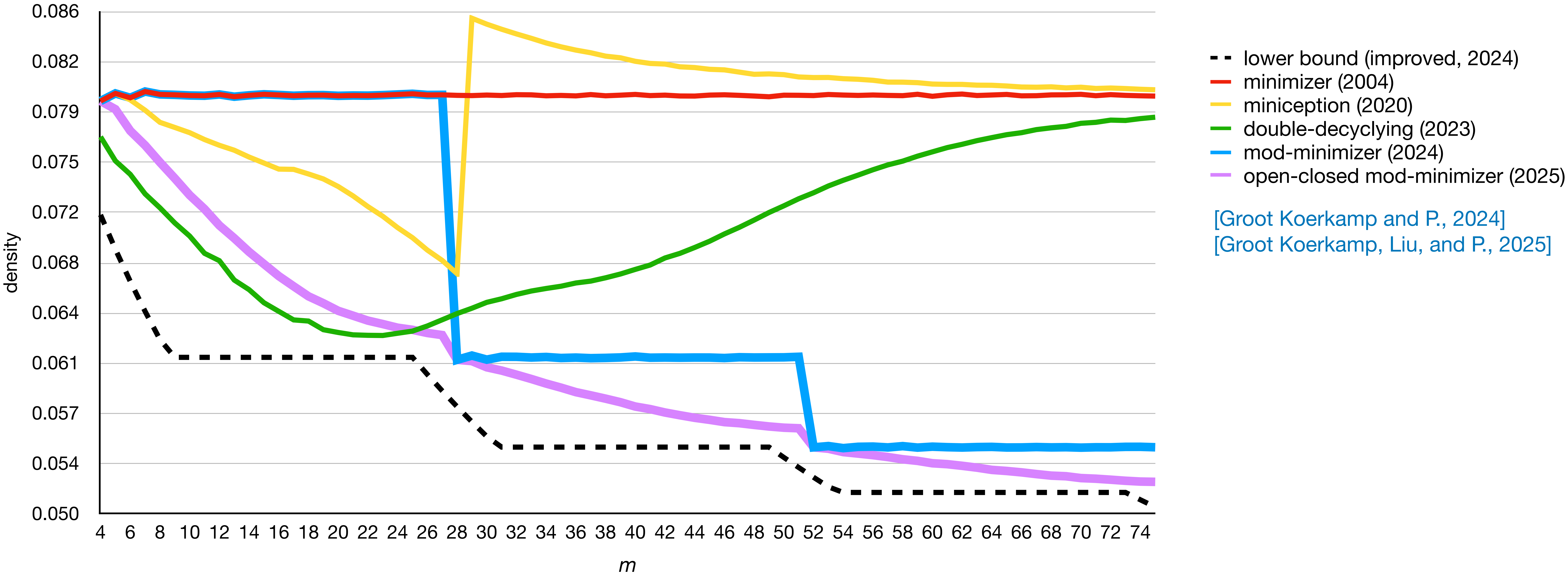
Density by varying *m*



[Groot Koerkamp and P., 2024]

- Example for $w = 24$, so $k = 24 + m - 1$.
- Measured over a string of 10 million i.i.d. random characters with an alphabet size of 4.
- https://github.com/jermp/minimizers

Density by varying *m*

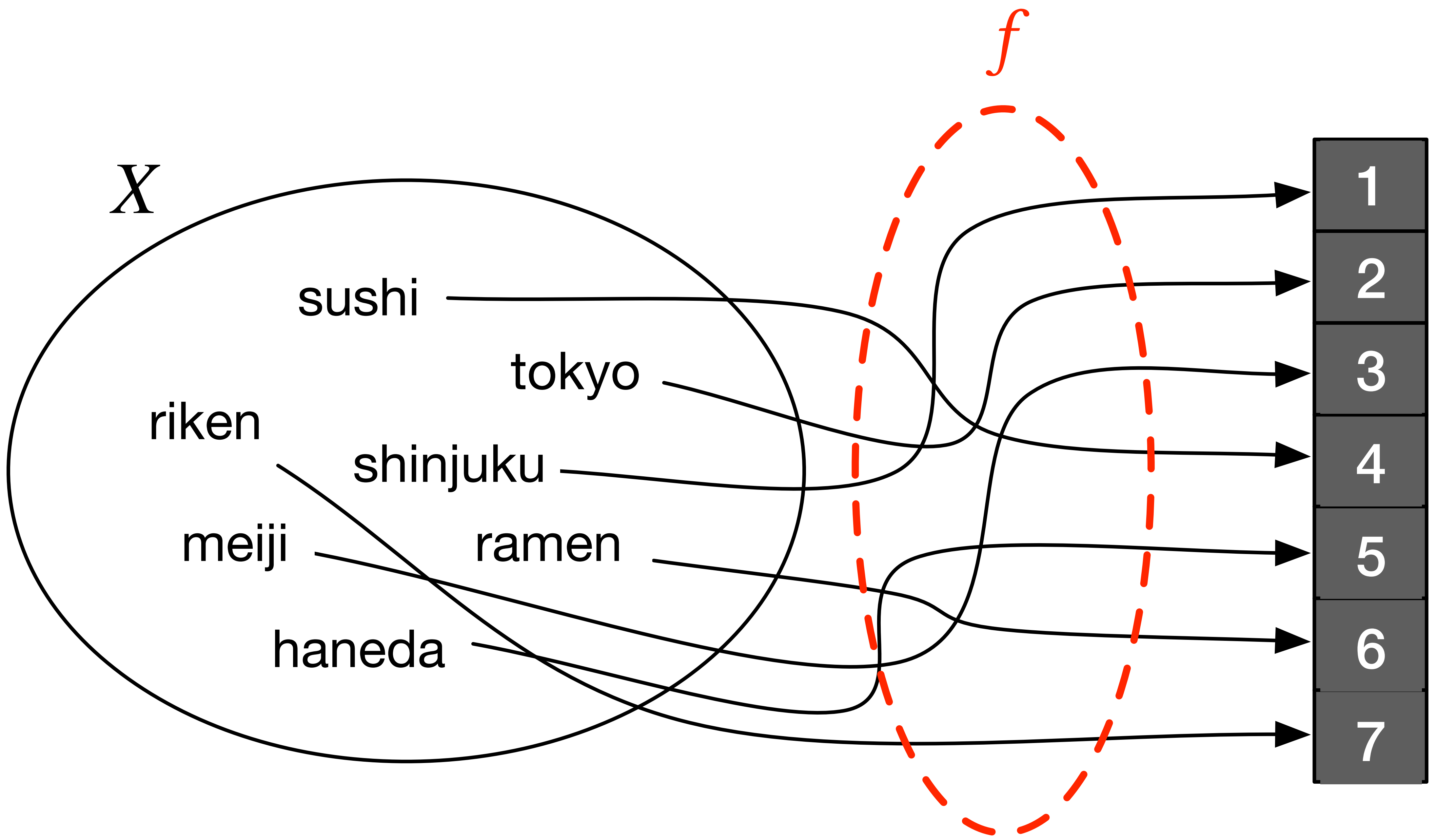


[Groot Koerkamp and P., 2024]
[Groot Koerkamp, Liu, and P., 2025]

- Example for $w = 24$, so $k = 24 + m - 1$.
- Measured over a string of 10 million i.i.d. random characters with an alphabet size of 4.
- https://github.com/jermp/minimizers

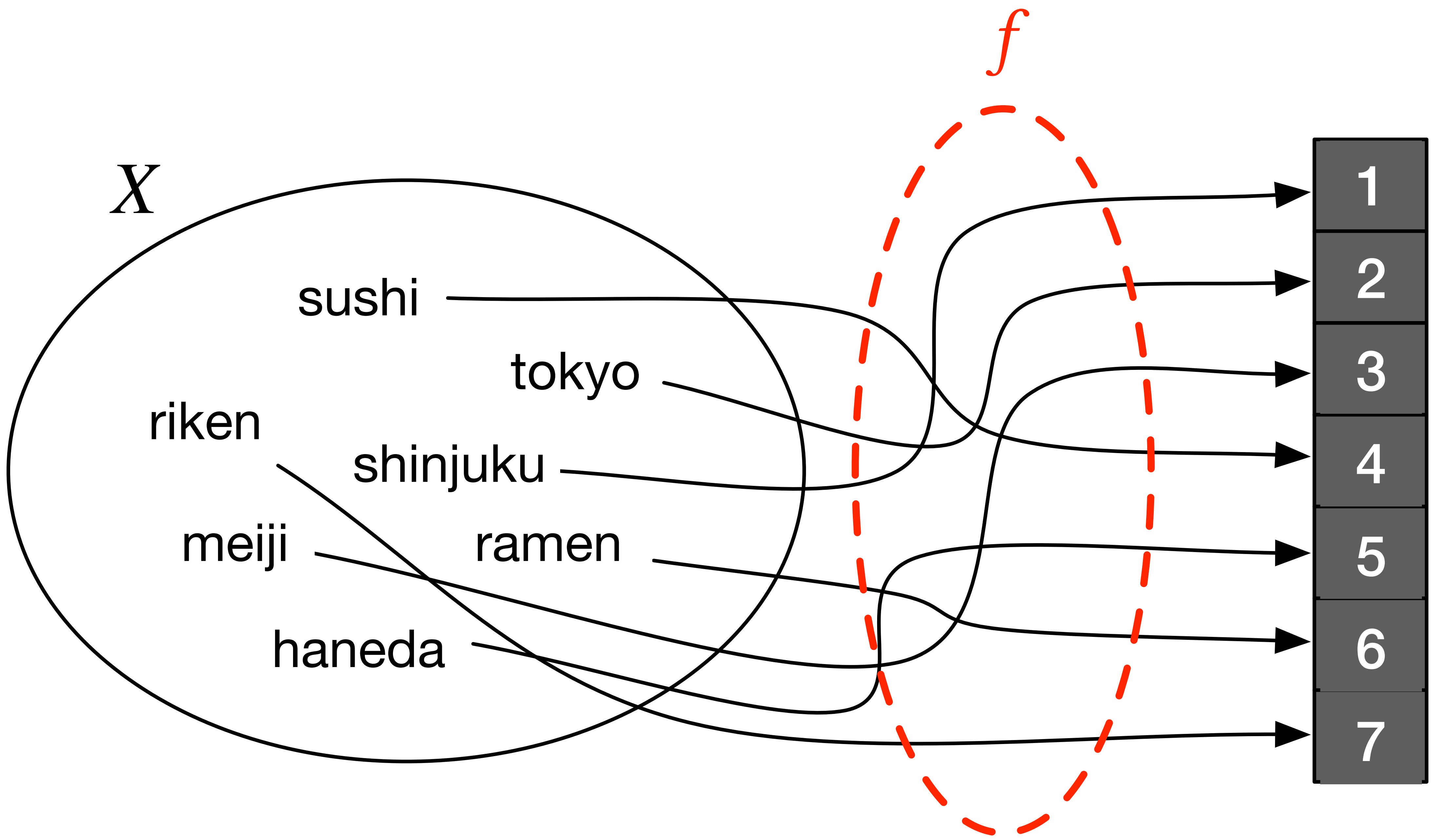
Minimal perfect hashing

MPHF. Given a set X of n distinct keys, a function f that *bijectively* maps the keys of X into the range $\{1,\ldots,n\}$ is called a *minimal perfect hash function* (MPHF) for X.



Minimal perfect hashing

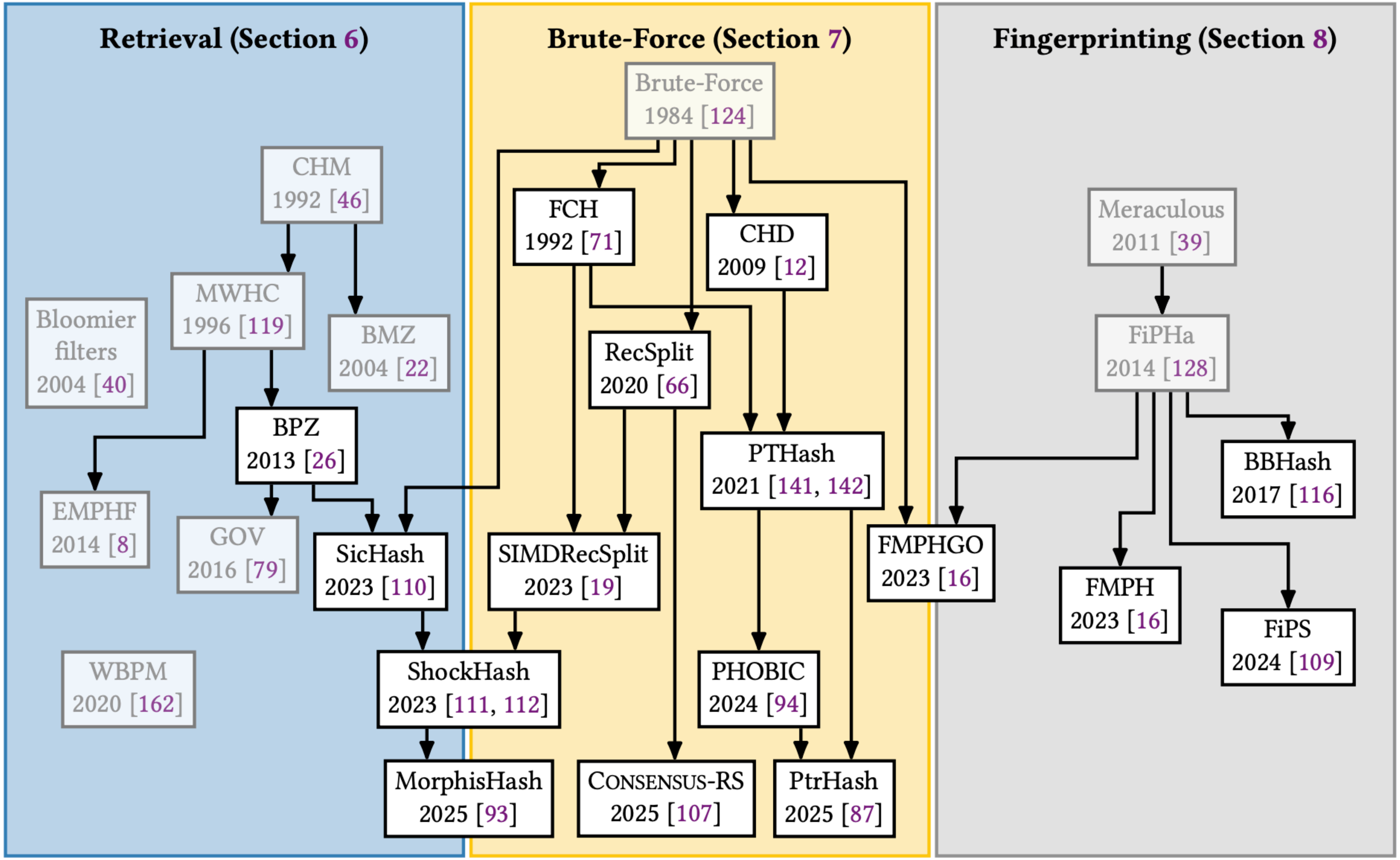
MPHF. Given a set X of n distinct keys, a function f that *bijectively* maps the keys of X into the range $\{1,\ldots,n\}$ is called a *minimal perfect hash function* (MPHF) for X.



- Space lower bound of $\log_2(e) \approx 1.443$ bits/key [Mehlhorn, 1982].
- Many approaches available (see next).
- Most of them have:
  - Constant-time evaluation.
  - Expected linear-time construction.
  - Take 1.8 — 3 bits/key.

Modern Minimal Perfect Hashing: A Survey
Lehmann, Mueller, P., Sanders, Vigna, Walzer, 2025
https://arxiv.org/pdf/2506.06536

Minimal perfect hashing



The PTHash “family”

- The **fastest** functions for lookup time: 30-50 ns/key.
- Also very fast to build and space-efficient: they take from 1.7 to 3.0 bits/key.
  - [P. and Trani, 2021, 2023]

    PTHash

    https://github.com/jermp/pthash
  - PHOBIC [Hermann et al., 2024]
  - PtrHash [Groot Koerkamp, 2025]
  - PHast [Beling and Sanders, 2025]

3. Sparse and skew hashing of k-mers

Super-k-mers

- **Property.** Consecutive k-mers are likely to have the same minimizer.

  Example for k=13 and m=4:

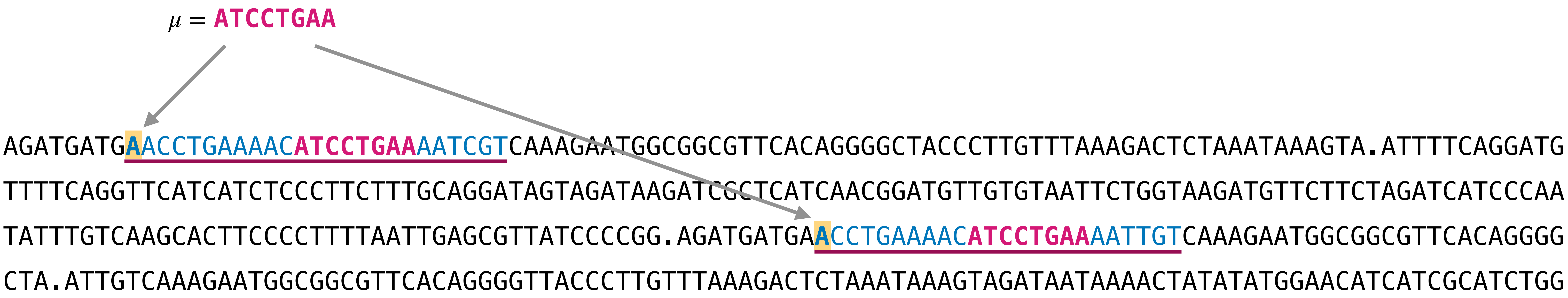
```
ACGGTAGAACCGATTCAAATTCGATCGATTAATTAGAGCGATAAC...
ACGGTAGAACCGA
 CGGTAGAACCGAT
  GGTAGAACCGATT
   GTAGAACCGATTC
    TAGAACCGATTCA
     AGAACCGATTCAA
      GAACCGATTCAAA
       AACCGATTCAAAT
        ...
```

  super-k-mer

- **Super-k-mer.** Given a string, a super-k-mer is a **maximal** sequence of consecutive k-mers having the same minimizer.

Sparse hashing

- Locate super-k-mers with an array of offsets into the strings, indexed by a **minimal perfect hash function** built on the set of minimizers.
- The space of this index is then proportional to the number of minimizers, that are sparse in the sequence (i.e., we expect to see $(k-m+2)/2$ times less *random* minimizers than k-mers).
- Upon Lookup(x):
  1. Compute the the minimizer of x, say μ;
  2. Locate and scan the "bucket" of μ — the set of super-k-mers that have minimizer μ.



$\mu =$ **ATCCTGAA**

AGATGATGAACCTGAAAAC**ATCCTGAA**AATCGTCAAAGAATGGCGGCGTTCACAGGGGCTACCCTTGTTTAAAGACTCTAAATAAAGTA.ATTTTCAGGATG
TTTTCAGGTTCATCATCTCCCTTCTTTGCAGGATAGTAGATAAGATCGCTCATCAACGGATGTTGTGTAATTCTGGTAAGATGTTCTTCTAGATCATCCCAA
TATTTGTCAAGCACTTCCCCTTTTAATTGAGCGTTATCCCCGG.AGATGATGAACCTGAAAAC**ATCCTGAA**AATTGTCAAAGAATGGCGGCGTTCACAGGGG
CTA.ATTGTCAAAGAATGGCGGCGTTCACAGGGGTTACCCTTGTTTAAAGACTCTAAATAAAGTAGATAATAAAACTATATATGGAACATCATCGCATCTGG

Sparse hashing — Example

A collection of **4** stitched unitigs:
285 k-mers for k=31, $N = 408$ bases in total

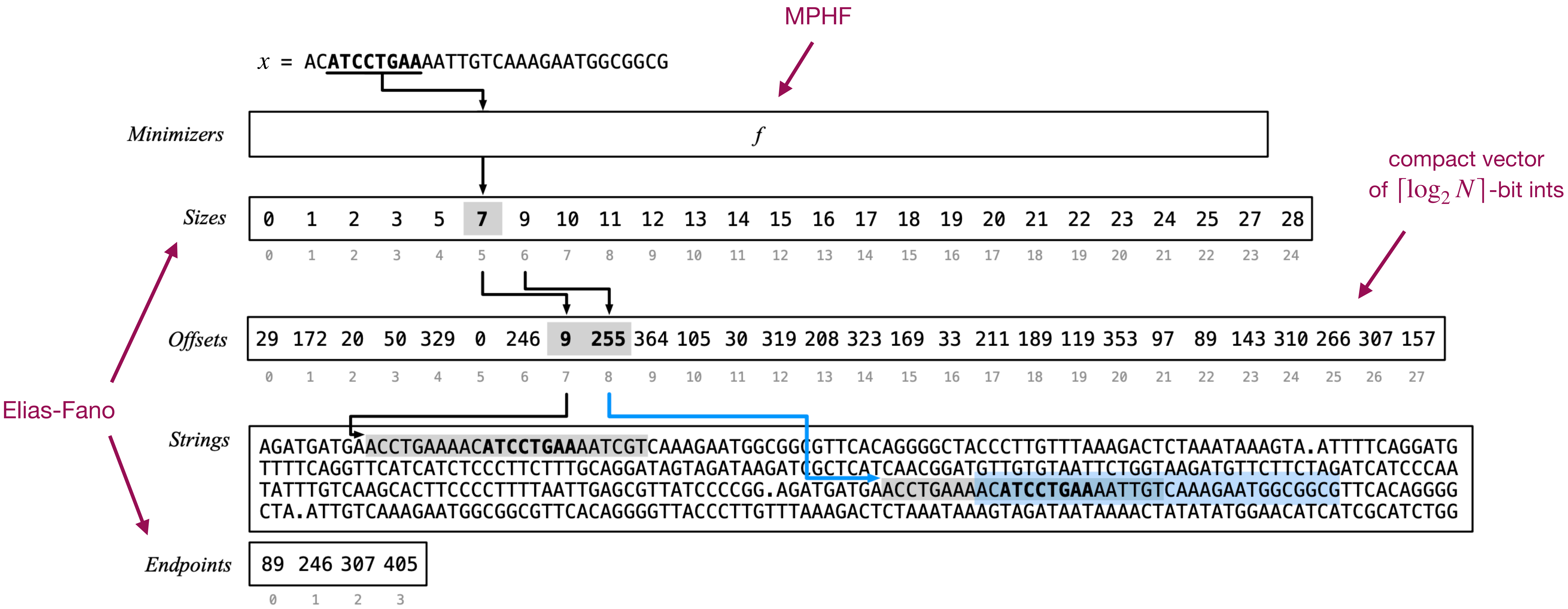
```
AGATGATGAACCTGAAAACATCCTGAAAATCGTCAAAGAATGGCGG
CGTTCACAGGGGCTACCCTTGTTTAAAGACTCTAAATAAAGTA.AT
TTTCAGGATGTTTTCAGGTTCATCATCTCCCTTCTTTGCAGGATAG
TAGATAAGATCGCTCATCAACGGATGTTGTGTAATTCTGGTAAGAT
GTTCTTCTAGATCATCCCAATATTTGTCAAGCACTTCCCCTTTTAA
TTGAGCGTTATCCCCGG.AGATGATGAACCTGAAAACATCCTGAAA
ATTGTCAAAGAATGGCGGCGTTCACAGGGGCTA.ATTGTCAAAGAA
TGGCGGCGTTCACAGGGGTTACCCTTGTTTAAAGACTCTAAATAAA
GTAGATAATAAAACTATATATGGAACATCATCGCATCTGG
```

24 minimizers, for $m = 8$

```
TCGTCAAA: 29
CATCCCAA: 172
ATCGTCAA: 20
GACTCTAA: 50 329
AACCTGAA: 0 246
ATCCTGAA: 9 255
GAACATCA: 364
GCAGGATA: 105
AGGGGCTA: 30
CTTGTTTA: 319
GAGCGTTA: 208
TTTAAAGA: 323
CTTCTAGA: 169
GGCTACCC: 33
CGTTATCC: 211
AGCACTTC: 189
AAGATCGC: 119
AACTATAT: 353
CCTTCTTT: 97
TTCAGGTT: 89
ACGGATGT: 143
ACAGGGGT: 310
TGTCAAAG: 266 307
TAATTCTG: 157
```

offsets

Data structure



MPHF

x = AC**ATCCTGAA**AATTGTCAAAGAATGGCGGCG

Minimizers f

Sizes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	1	2	3	5	**7**	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	27	28

compact vector of $\lceil \log_2 N \rceil$-bit ints

Offsets

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
29	172	20	50	329	0	246	**9**	**255**	364	105	30	319	208	323	169	33	211	189	119	353	97	89	143	310	266	307	157

Elias-Fano

Strings

AGATGATGAACCTGAAAAC**ATCCTGAA**AATCGTCAAAGAATGGCGGCGTTCACAGGGGCTACCCTTGTTTAAAGACTCTAAATAAAGTA.ATTTTCAGGATG
TTTTCAGGTTCATCATCTCCCTTCTTTGCAGGATAGTAGATAAGATCGCTCATCAACGGATGTTGTGTAATTCTGGTAAGATGTTCTTCTAGATCATCCCAA
TATTTGTCAAGCACTTCCCCTTTTAATTGAGCGTTATCCCCGG.AGATGATGAACCTGAAAAC**ATCCTGAA**AATTGTCAAAGAATGGCGGCGTTCACAGGGG
CTA.ATTGTCAAAGAATGGCGGCGTTCACAGGGGTTACCCTTGTTTAAAGACTCTAAATAAAGTAGATAATAAAACTATATATGGAACATCATCGCATCTGG

Endpoints

0	1	2	3
89	246	307	405

The order of the k-mers in the SPSS is preserved

- **Order-Preserving Property.** If $x[2..k] = y[1..k-1]$, i.e., y is the "successor" of x, then: Lookup(y) = Lookup$(x) + 1$.

- So the hash code i = Lookup(x) can be directly used to **associate** some satellite information to the k-mer x, e.g., its abundance, color set, etc.

- Any order on the strings of SPSS(S) uniquely determines an order $i = 1,\ldots,n$ for the k-mers $x_i \in$ SPSS(S), thus: Lookup$(x_i) = i$.

The order of the k-mers in the SPSS is preserved

- **Order-Preserving Property.** If $x[2..k] = y[1..k-1]$, i.e., y is the "successor" of x, then: Lookup(y) $=$ Lookup(x) $+ 1$.

- So the hash code $i =$ Lookup(x) can be directly used to **associate** some satellite information to the k-mer x, e.g., its abundance, color set, etc.

- Any order on the strings of SPSS(S) uniquely determines an order $i = 1,\ldots,n$ for the k-mers $x_i \in$ SPSS(S), thus: Lookup(x_i) $= i$.

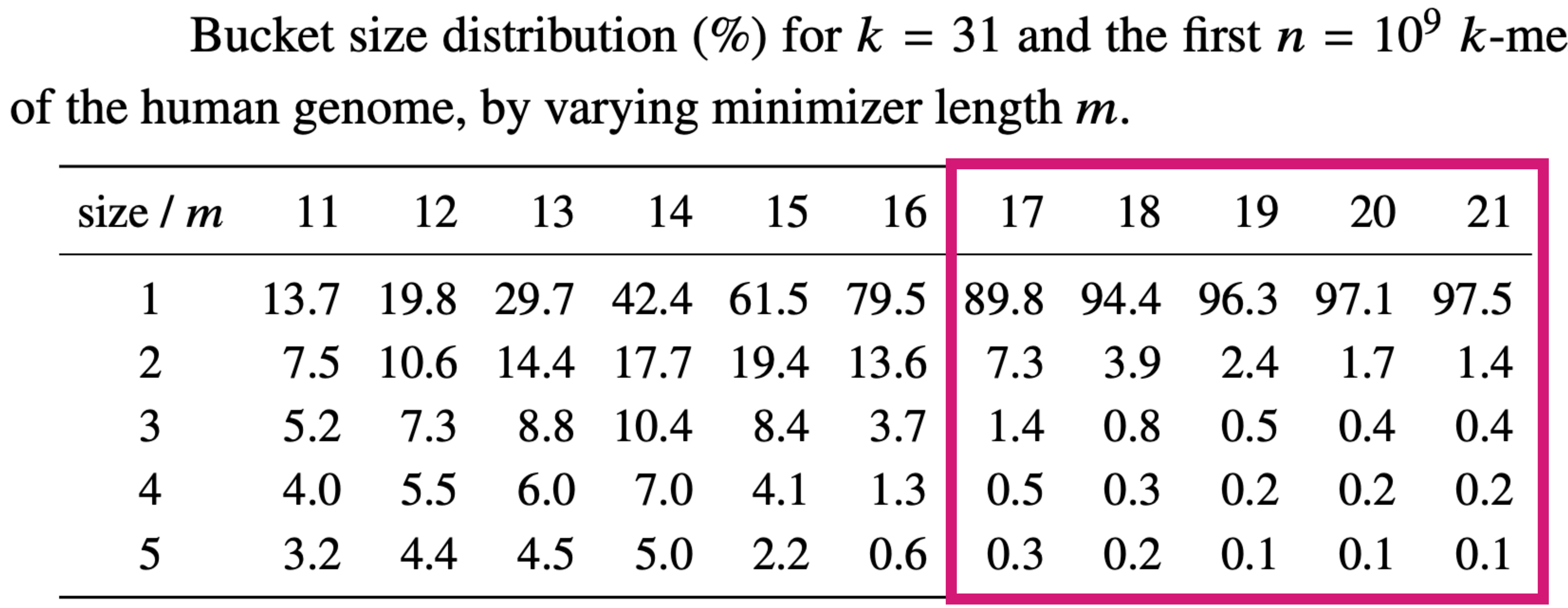
- This property makes **compression of satellite information** easy and effective.
  We will see another example on **4th July**.

Skew hashing

- **Problem.** Some buckets can be very large.

For example on the human genome (GRCh38), for $k = 31$ and $m = 20$: largest bucket size can be as large as 3.6×10^4.

- **Property.** Minimizers have a (very) **skew** distribution for sufficiently-long length m.



Bucket size distribution (%) for $k = 31$ and the first $n = 10^9$ k-mers of the human genome, by varying minimizer length m.

size / m	11	12	13	14	15	16	17	18	19	20	21
1	13.7	19.8	29.7	42.4	61.5	79.5	89.8	94.4	96.3	97.1	97.5
2	7.5	10.6	14.4	17.7	19.4	13.6	7.3	3.9	2.4	1.7	1.4
3	5.2	7.3	8.8	10.4	8.4	3.7	1.4	0.8	0.5	0.4	0.4
4	4.0	5.5	6.0	7.0	4.1	1.3	0.5	0.3	0.2	0.2	0.2
5	3.2	4.4	4.5	5.0	2.2	0.6	0.3	0.2	0.1	0.1	0.1

On the **full** human genome (GRCh38),
for $k = 31$ and $m = 20$:
`2,505,445,761` k-mers
`421,845,806` minimizers
`388,018,280` (91.98%) only appear **once**!

Skew hashing

- We fix an integer ℓ: by virtue of the skew distribution, the fraction of buckets having **more than** 2^ℓ super-k-mers is **small**.
- So, we can afford a MPHF over the set of k-mers that belong to such super-k-mers. The output of the MPHF for a k-mer x is the **identifier** of the super-k-mer where x is present.
- Upon Lookup, we will scan **one** super-k-mer only.

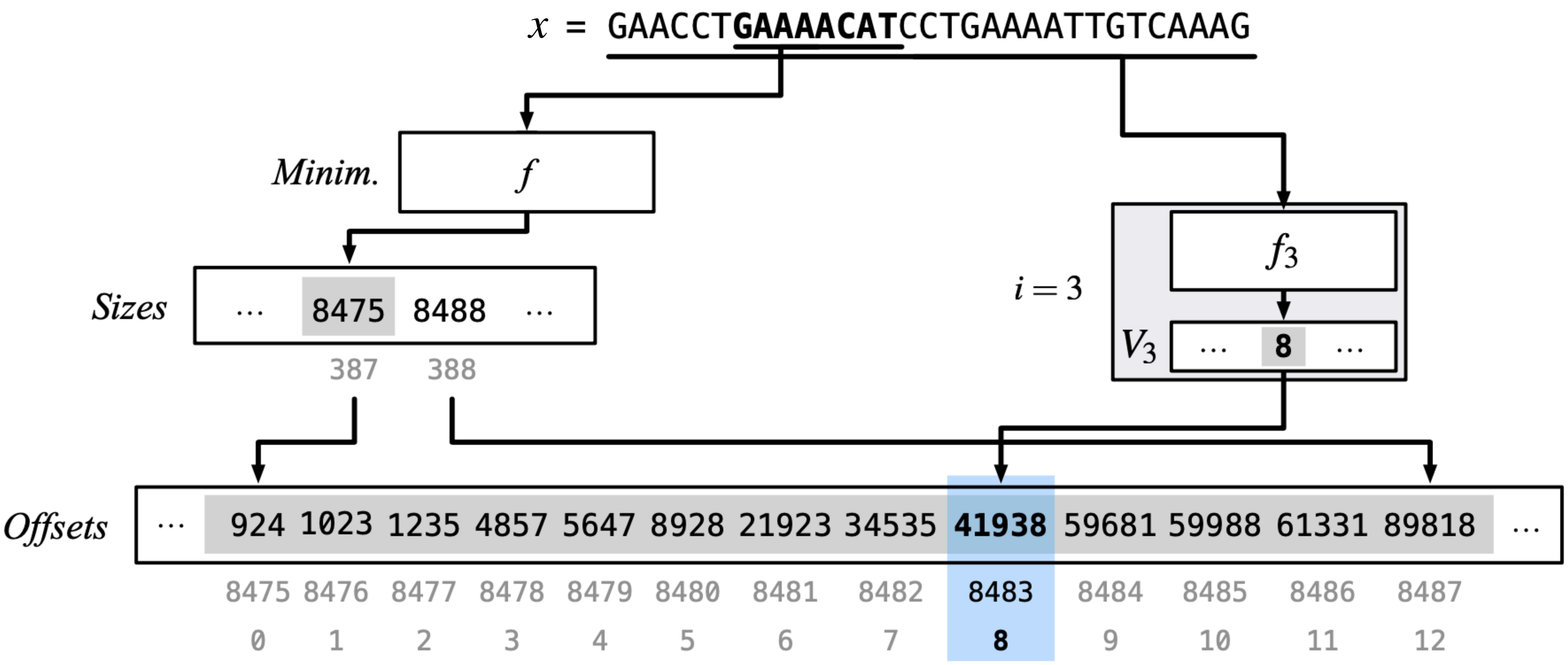
Bucket size distribution (%) for $k = 31$ and the first $n = 10^9$ k-mers of the human genome, by varying minimizer length m.

size / m	11	12	13	14	15	16	17	18	19	20	21
1	13.7	19.8	29.7	42.4	61.5	79.5	89.8	94.4	96.3	97.1	97.5
2	7.5	10.6	14.4	17.7	19.4	13.6	7.3	3.9	2.4	1.7	1.4
3	5.2	7.3	8.8	10.4	8.4	3.7	1.4	0.8	0.5	0.4	0.4
4	4.0	5.5	6.0	7.0	4.1	1.3	0.5	0.3	0.2	0.2	0.2
5	3.2	4.4	4.5	5.0	2.2	0.6	0.3	0.2	0.1	0.1	0.1

For $\ell = 2$, just
100.0 – (97.1 + 1.7 + 0.4 + 0.2)% = 0.6% of
buckets with more than $2^{\ell=2} = 4$ super-k-mers.

Skew hashing — Example

Example for $\ell = 3$.



Implementation and results

- These ideas have been implemented in a software tool (C++17):

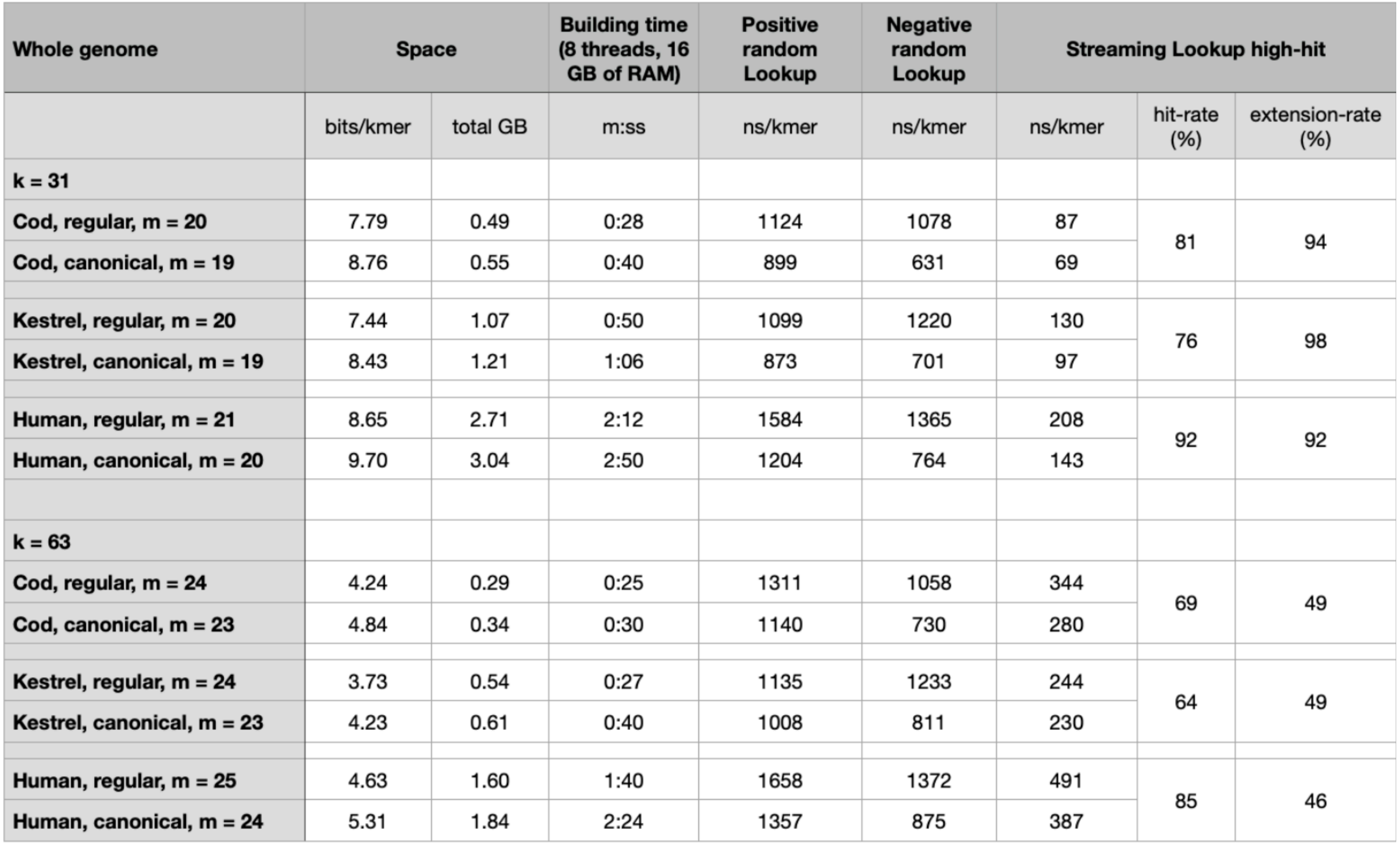
SSHash

https://github.com/jermp/sshash

New benchmarks

https://github.com/jermp/sshash/tree/master/benchmarks

Ubuntu 18.04.6; gcc 9.4; Intel Xeon W-2245 CPU @ 3.90GHz; Results taken on 18/06/2025



Whole genome	**Space**		**Building time (8 threads, 16 GB of RAM)**	**Positive random Lookup**	**Negative random Lookup**	**Streaming Lookup high-hit**		
	bits/kmer	total GB	m:ss	ns/kmer	ns/kmer	ns/kmer	hit-rate (%)	extension-rate (%)
k = 31								
Cod, regular, m = 20	7.79	0.49	0:28	1124	1078	87	81	94
Cod, canonical, m = 19	8.76	0.55	0:40	899	631	69		
Kestrel, regular, m = 20	7.44	1.07	0:50	1099	1220	130	76	98
Kestrel, canonical, m = 19	8.43	1.21	1:06	873	701	97		
Human, regular, m = 21	8.65	2.71	2:12	1584	1365	208	92	92
Human, canonical, m = 20	9.70	3.04	2:50	1204	764	143		
k = 63								
Cod, regular, m = 24	4.24	0.29	0:25	1311	1058	344	69	49
Cod, canonical, m = 23	4.84	0.34	0:30	1140	730	280		
Kestrel, regular, m = 24	3.73	0.54	0:27	1135	1233	244	64	49
Kestrel, canonical, m = 23	4.23	0.61	0:40	1008	811	230		
Human, regular, m = 25	4.63	1.60	1:40	1658	1372	491	85	46
Human, canonical, m = 24	5.31	1.84	2:24	1357	875	387		

To sum up

- SSHash is an **order-preserving** k-mer dictionary.

- Three important tools:
  1. spectrum-preserving string sets;
  2. minimizers;
  3. minimal perfect hashing.

- Ingredients:
  - **Sparse indexing** to obtain good space effectiveness;
  - **Skew hashing** to guarantee fast lookup for “heavy” buckets.

- Code in C++17 is available at: https://github.com/jermp/sshash.

Extensions

- k-mer **abundances** [P. 2022, 2023]
- **sequence membership**: a sequence S is considered as present in the dictionary if at least a given fraction of its k-mers is found in the dictionary [Schmidt, Khan, Alanko, P., Tomescu, 2023]
- reference indexing: store also **positional** information for each k-mer [Fan, Khan, P., Patro, 2023]
- **colored de Bruijn graphs**: annotate each k-mer with the set of its “colors” (i.e., the references where it appears) [Fan, Khan, Singh, P., Patro, 2023, 2024; Fan, P., Patro, 2024; Campanelli, P., Fan, Patro, 2024; Campanelli, P., Patro, 2025]

Extensions

- k-mer **abundances** [P. 2022, 2023]
- **sequence membership**: a sequence S is considered as present in the dictionary if at least a given fraction of its k-mers is found in the dictionary [Schmidt, Khan, Alanko, P., Tomescu, 2023]
- reference indexing: store also **positional** information for each k-mer [Fan, Khan, P., Patro, 2023]
- **colored de Bruijn graphs**: annotate each k-mer with the set of its “colors” (i.e., the references where it appears) [Fan, Khan, Singh, P., Patro, 2023, 2024; Fan, P., Patro, 2024; Campanelli, P., Fan, Patro, 2024; Campanelli, P., Patro, 2025]

4th July, at 13:50